OGC API – Features

Winter School 2022

Ms. Prajwalita J. Chavan
IIT, Bombay

Overview

- About OGC API – Features
- Resources
- Introduction
- Encoding
- Requirements classes ‘core’
- API landing page
- API Definition
- Conformance declaration
- Feature Collection
- Features
- Feature
- Requirements classes for encodings
- Requirements classes for ‘HTML’
- Requirement Class “GeoJSON’
- Requirement Class “GML”
- Requirement Class "Map Background"
- Security

OGC API - Features

- **Publication Date:** 2022-05-11
- OGC API Features provides API building blocks to **create, modify and query features** on the Web.
- the OGC API Features standards offer **direct, fine-grained access to the data at the feature (object) level.**
- This standard specifies **discovery and query operations** that are implemented using the HTTP GET method.
- **WFS** uses a Remote-Procedure-Callover-HTTP architectural style using XML for any payloads.

Resources

Resource	Path	HTTP method	Document reference
Landing page	/	GET	7.2 API landing page
Conformance declaration	/conformance	GET	7.4 Declaration of conformance classes
Feature collections	/collections	GET	7.13 Feature collections
Feature collection	/collections/{collectionId}	GET	7.14 Feature collection
Features	/collections/{collectionId}/items	GET	7.15 Features
Feature	/collections/{collectionId}/items/{featureId}	GET	7.16 Feature

Introduction

1. Main requirements class: **Core**
2. requirements class: **CRS**
3. requirements class: **Filtering**
4. requirements class: **CRUD**

1. Main requirements class: Core

The ***Core*** does not mandate a specific **encoding** or format for representing features or feature collections:

- HTML
- GeoJSON
- Geography Markup Language (GML), Simple Features Profile, Level 0
- Geography Markup Language (GML), Simple Features Profile, Level 2
- OpenAPI Specification 3.0
- Other encoding

Encodings

- **HTML** is the core language of the World Wide Web. A server that supports HTML will support browsing the data with a web browser
- **GeoJSON** is a commonly used format that is simple to understand and well supported by tools and software libraries.
- **GML** supports more complex requirements than GeoJSON. GML is more complex to handle for both servers and clients.

Requirements Class "Core"

- The entry point is a **Landing page (path /)**

- The **Landing page provides** links to:
  - the **API definition** (link relations service-desc and service-doc),
  - the **Conformance declaration** (path /conformance, link relation conformance), and
  - the **Collections** (path /collections, link relation data)

  (YAML: writing configuration file)

Requirement 1	**/req/core/root-op**
A	The server SHALL support the HTTP GET operation at the path /.

API landing page

Requirement 2	/req/core/root-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema landingPage.yaml and include at least links to the following resources: • the API definition (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'conformance') • `/collections` (relation type 'data')

Schema for Landing Page

```
type: object
required:
  - links
properties:
  title:
    type: string
  description:
    type: string
  links:
    type: array
    items:
      $ref:
http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml
```

Landing Page Response Document


```
{
  "title": "Buildings in Bonn",
  "description": "Access to data about buildings in the city of Bonn via a Web API
that conforms to the OGC API Features specification.",
  "links": [
    { "href": "http://data.example.org/",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/api",
      "rel": "service-desc", "type":
"application/vnd.oai.openapi+json;version=3.0", "title": "the API definition" },
    { "href": "http://data.example.org/api.html",
      "rel": "service-doc", "type": "text/html", "title": "the API documentation"
},
    { "href": "http://data.example.org/conformance",
      "rel": "conformance", "type": "application/json", "title": "OGC API
conformance classes implemented by this server" },
    { "href": "http://data.example.org/collections",
      "rel": "data", "type": "application/json", "title": "Information about the
feature collections" }
  ]
}
```

API definition

Requirement 3	/req/core/api-definition-op
A	The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.

Permission 1	/per/core/api-definition-uri
A	The API definition is metadata about the API and strictly not part of the API itself, but it MAY be hosted as a sub-resource to the base path of the API, for example, at path `/api`. There is no need to include the path of the API definition in the API definition itself.

Requirement 4	/req/core/api-definition-success
A	A GET request to the URI of an API definition linked from the landing page (link relations `service-desc` or `service-doc`) with an `Accept` header with the value of the link property `type` SHALL return a document consistent with the requested media type.

Conformance declaration

Requirement 5	/req/core/conformance-op
A	The server SHALL support the HTTP GET operation at the path `/conformance`.

Requirement 6	/req/core/conformance-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema confClasses.yaml and list all OGC API conformance classes that the server conforms to.

Feature Collections

Requirement 11	/req/core/fc-md-op
A	The server SHALL support the HTTP GET operation at the path `/collections`.

Requirement 12	/req/core/fc-md-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be based upon the OpenAPI 3.0 schema collections.yaml.

Feature Collections

Requirement 17	/req/core/sfc-md-op
A	The server SHALL support the HTTP GET operation at the path `/collections/{collectionId}`.
B	The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`).

Requirement 18	/req/core/sfc-md-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.
B	The content of that response SHALL be consistent with the content for this feature collection in the `/collections` response. That is, the values for `id`, `title`, `description` and `extent` SHALL be identical.

Features

Requirement 19	/req/core/fc-op
A	For every feature collection identified in the feature collections response (path `/collections`), the server SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items`.
B	The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`).

Requirement 21	/req/core/fc-limit-response-1
A	The response SHALL not contain more features than specified by the optional `limit` parameter. If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more features than this maximum value.
B	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

Feature

Requirement 32	/req/core/f-op
A	For every feature in a feature collection (path `/collections/{collectionId}`), the server SHALL support the HTTP GET operation at the path `/collections/{collectionId}/items/{featureId}`.
B	The parameter `collectionId` is each `id` property in the feature collections response (JSONPath: `$.collections[*].id`). `featureId` is a local identifier of the feature.

Requirements Class "HTML"

Requirement 35	/req/html/definition
A	Every `200`-response of an operation of the server SHALL support the media type `text/html`.

Requirement 36	/req/html/content
A	Every `200`-response of the server with the media type `text/html` SHALL be a HTML 5 document that includes the following information in the HTML body: • all information identified in the schemas of the Response Object in the HTML `<body>`, and • all links in HTML `<a>` elements in the HTML `<body>`.

Requirements Class "GeoJSON"

Requirement 37	/req/geojson/definition
A	200-responses of the server SHALL support the following media types: • application/geo+json for resources that include feature content, and • application/json for all other resources.

Requirements Class "Geography Markup Language (GML), Simple Features Profile, Level 0"

Requirement 39	/req/gmlsf0/definition
A	`200`-responses of the server SHALL support the following media types: • `application/gml+xml; version=3.2; profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` for resources that include feature content, • `application/xml` for all other resources.

Requirement Class "Map Background"

Resource	Path	XML root element
Landing page	/	core:LandingPage
Conformance declaration	/conformance	core:ConformsTo
Feature collections	/collections	core:Collections
Feature collection	/collections/{collectionId}	core:Collections, with just one entry for the collection collectionId
Features	/collections/{collectionId}/items	sf:FeatureCollection
Feature	/collections/{collectionId}/items/{featureId}	substitutable for gml:AbstractFeature

Security

- A valuable resource is the **Common Weakness Enumeration (CWE)** registry

- The CWE is organized around three views
  - **Research**: facilitates **research into weaknesses** and can be leveraged to systematically identify theoretical gaps within CWE.

  - **Architectural**: organizes weaknesses according to common architectural security tactics. It is intended to assist architects in **identifying potential mistakes** that can be made when designing software.

  - **Development**: **organizes weaknesses around concepts** that are frequently used or encountered in software development.

Security: Multiple Servers

- The **implementation of an API** may span a number of servers.
- Each server is an **entry point** into the API.
- Without careful management, **information** which is not **accessible** though one server may be accessible through another

Path Manipulation

- A **transaction operation adds new or updates existing resources** on the API. This capability provides a whole new set of tools to an attacker.
- **GET: Validate** all GET **URLs** to make sure they are not trying to access resources they should not have access to.
- **PUT and POST:** APIs which support transaction operations should validate that an update does **not contain any malignant content** prior to exposing it through the API.

2. Requirement Class for CRS

Requirement 1	/req/crs/crs-uri
Each CRS supported by a server SHALL be referenceable by a uniform resource identifier (i.e. a URI).	

Recommendation 1	/rec/crs/crs-format-model

Servers that implement this extension SHOULD be able to recognize and generate CRS identifiers with the following format model:

`http://www.opengis.net/def/crs/{authority}/{version}/{code}`

In this format model, the token `{authority}` is a placeholder for a value that designates to authority responsible for the definition of this CRS. Typical values include "EPSG" and "OGC".

The token `{version}` is a placeholder for the specific version of the CRS definition or `0` for un-versioned CRS definitions.

The token `{code}` is a placeholder for the authority's code for the CRS.

Requirement Class for global list of CRS identifiers


```
{
  "links": [
    { "href": "http://data.example.org/collections.json",
      "rel": "self", "type": "application/json", "title": "this document" },
    { "href": "http://data.example.org/collections.html",
      "rel": "alternate", "type": "text/html", "title": "this document as HTML" },
    { "href": "http://schemas.example.org/1.0/buildings.xsd",
      "rel": "describedby", "type": "application/xml", "title": "GML application schema for
Acme Corporation building data" },
    { "href": "http://download.example.org/buildings.gpkg",
      "rel": "enclosure", "type": "application/geopackage+sqlite3", "title": "Bulk download
(GeoPackage)", "length": 472546 }
  ],
  "crs": [
     "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
     "http://www.opengis.net/def/crs/EPSG/0/4326",
     "http://www.opengis.net/def/crs/EPSG/0/3857",
     "http://www.opengis.net/def/crs/EPSG/0/3395"
  ],
  "collections": [
     {
       "id": "bonn_buildings",
       "title": "Bonn Buildings",
       "description": "Buildings in the city of Bonn.",
       "extent": {
         "spatial": {
           "bbox": [ [ 7.01, 50.63, 7.22, 50.78 ] ]
         },
```

Output format considerations

HTML: The **HyperText Markup Language** or **HTML** is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets(**CSS**) and scripting languages such as **JavaScript**. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Output format considerations

- **XML: Extensible Markup Language** (**XML**) is a markup language and file format **for storing, transmitting, and reconstructing arbitrary data**
- It defines a **set of rules** for encoding documents in a format that is both human-readable and machine-readable

```
<?xml version="1.0" encoding="UTF-8"?>
<fruits>
    <item>
        <id>1000</id>
        <name>Apple</name>
        <price>4</price>
        <quantity>133</quantity>
    </item>
</fruits>
```

Output format considerations

- **GML:** GML or Geography Markup Language is an XML based encoding standard for geographic information **developed by the OpenGIS Consortium (OGC).** GML is concerned with the **representation of the geographic data** content. Of course we can also use GML to make maps. Like any XML encoding, GML **represents geographic information** in the form of text.

```
<Feature  fid="142" featureType="school"  Description="A middle school">
    <Polygon name="extent" srsName="epsg:27354">
      <LineString name="extent" srsName="epsg:27354">
        <CData>
          491888.999999459,5458045.99963358 491904.999999458,5458044.99963358
          491908.999999462,5458064.99963358 491924.999999461,5458064.99963358
          491925.999999462,5458079.99963359 491977.999999466,5458120.9996336
          491953.999999466,5458017.99963357 </CData>
      </LineString>
    </Polygon>
</Feature>
```

Output format considerations

- **YAML:**
- originally **Yet Another Markup Language**
- Now **YAML Ain't Markup Language**
- It is a **human-readable data-serialization language**
- It is commonly used for **configuration files** and in applications where **data** is being **stored or transmitted**
- YAML targets many of the same communications applications as Extensible Markup Language (XML) but has a **minimal syntax** which intentionally differs from Standard Generalized Markup Language (SGML)
- It uses both **Python-style indentation** to indicate **nesting**, and a more compact format

Output format considerations

```
---
receipt:      Oz-Ware Purchase Invoice
date:         2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:    A4786
      descrip:    Water Bucket (Filled)
      price:      1.47
      quantity:   4

    - part_no:    E1628
      descrip:    High Heeled "Ruby" Slippers
      size:       8
      price:      133.7
      quantity:   1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS
```

Output format considerations

- **JSON:** JavaScript Object Notation (JSON) has been gaining in popularity for **encoding data** in Web-based applications.
- JSON consists of sets of objects described by **name/value pairs**.
- JSON is **human readable and easily parseable**. However, JSON is **schemaless**.
- JSON and GeoJSON documents do **not include** an explicit definition of the **structure of the JSON objects** contained in them. Therefore, this standard is based on a normative JSON-LD context which allows each property to be explicitly defined as a URI. Furthermore, the JSON encoding is defined using JSON Schema which allows **validation of instances against these schemas**.

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Output format considerations

- **GeoJSON:**
- GeoJSON is a **format for encoding a variety of geographic data** structures. GeoJSON supports the following geometry types: **Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon.** Geometric objects with additional properties are Feature objects. Sets of features are contained by FeatureCollection objects.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

3. Filtering

- **Filter**: Filtering Expresssion
- **Filter-lang**: Filtering Language
- **Filter-crs**: CRS for Filtering

Filter: Filtering Expresssion

The Filter requirements class defines a general **parameter, filter**, whose value is a filter expression to be applied when retrieving resources. This is necessary to determine which resources should be included in a result set.

Requirement 4	**/req/filter/filter-param**

A	The HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) SHALL support a parameter `filter` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: filter`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

•**Filter-lang**: Filtering Language

Any **predicate language that can be suitably expressed as the value of an HTTP query parameter** may be specified as the value of the filter parameter. In order to specify that specific language that is being used, this clause defines

Requirement 5	**/req/filter/filter-lang-param**

A	The HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) SHALL support a parameter `filter-lang` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	YAML
	`name: filter-lang`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  enum:`
	`  - 'cql2-text'`
	`  - 'cql2-json'`
	`  default: 'cql2-text'`
	`style: form`

•**Filter-crs**: CRS for Filtering

Its parameter defined in this clause allows clients to assert which CRS is being **used to encode geometric values in a filter expression**.

Requirement 6	/req/filter/filter-crs-wgs84
A	If a HTTP GET operation on the path that fetches resource instances (e.g. `/collections/{collectionId}/items`) includes a `filter` parameter, but no `filter-crs` parameter, the server SHALL process all geometries in the filter expression using CRS84 (for coordinates without height) or CRS84h (for coordinates with ellipsoidal height) as the coordinate reference system (CRS).

4. CRUD

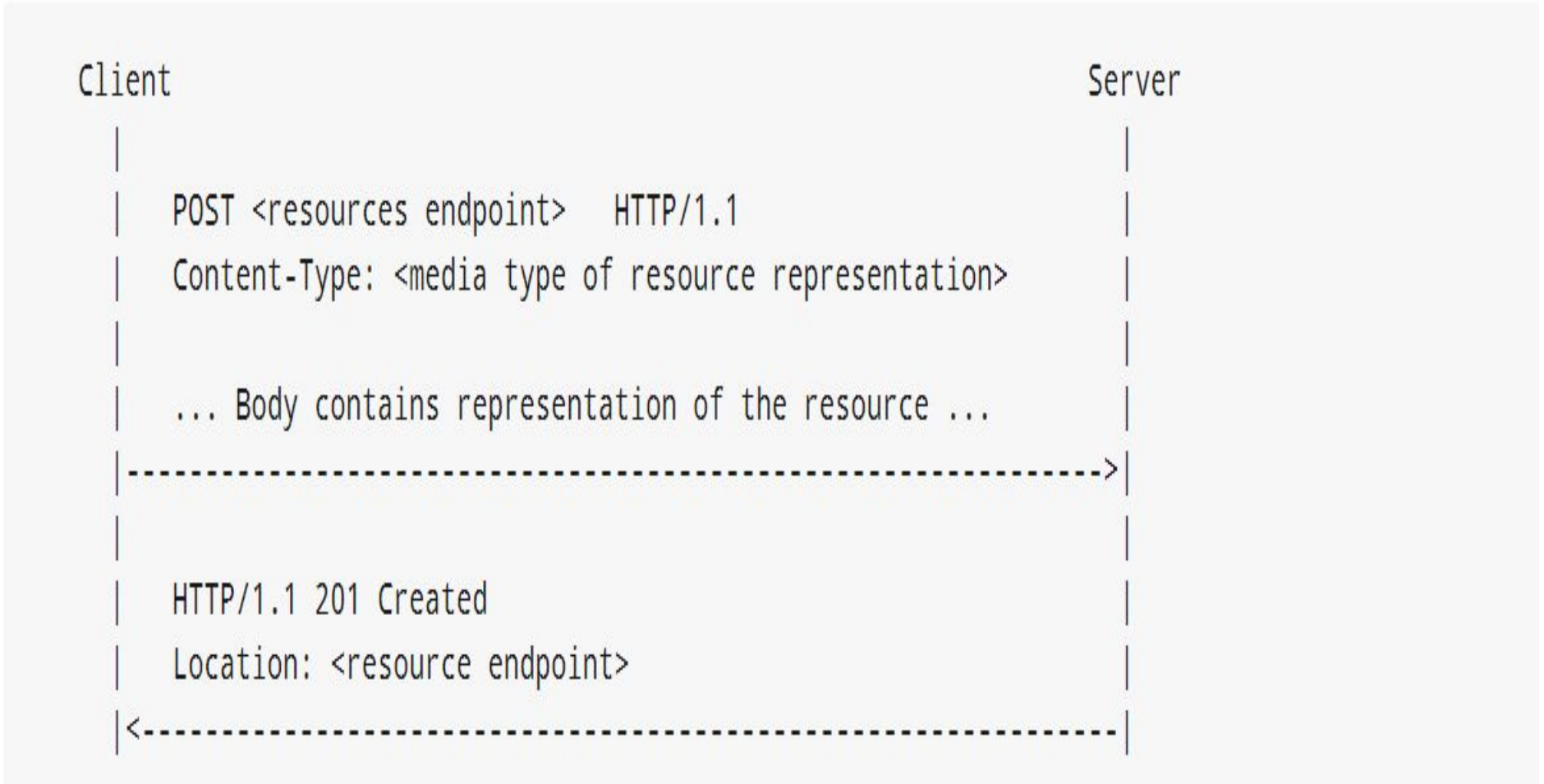
Resource endpoint	HTTP method			
	POST	PUT	PATCH	DELETE
/collections/{collectionId}/items	create	n/a	n/a	n/a
/collections/{collectionId}/items/{resourceId}	n/a	replace	update	delete

Requirements Class "Create/Replace/Delete"

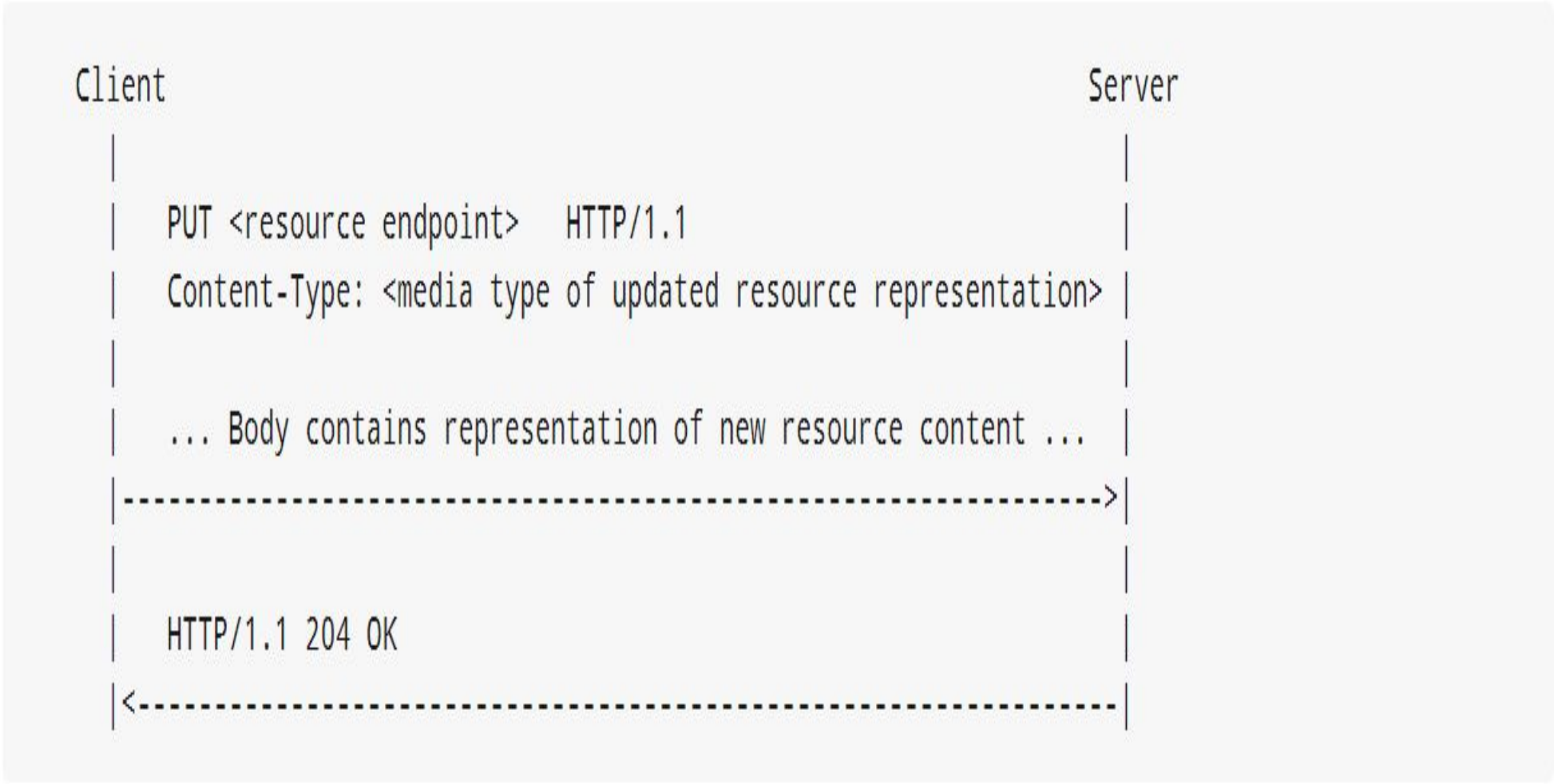
The HTTP **POST method is used to add** a new resource instance to a collection.
The HTTP **PUT method is used to replace** an existing resource in a collection with a replacement resource with the same resource identifier.
Finally, the HTTP **DELETE method is used to remove a resource** from a collection and **PATCH method to update**

Requirement 1	/req/core/methods
A	A server SHALL implement one or more of the methods HTTP POST, PUT and/or DELETE for each resource.
B	A server SHALL declare which methods are supported for each resource via the HTTP OPTIONS method.

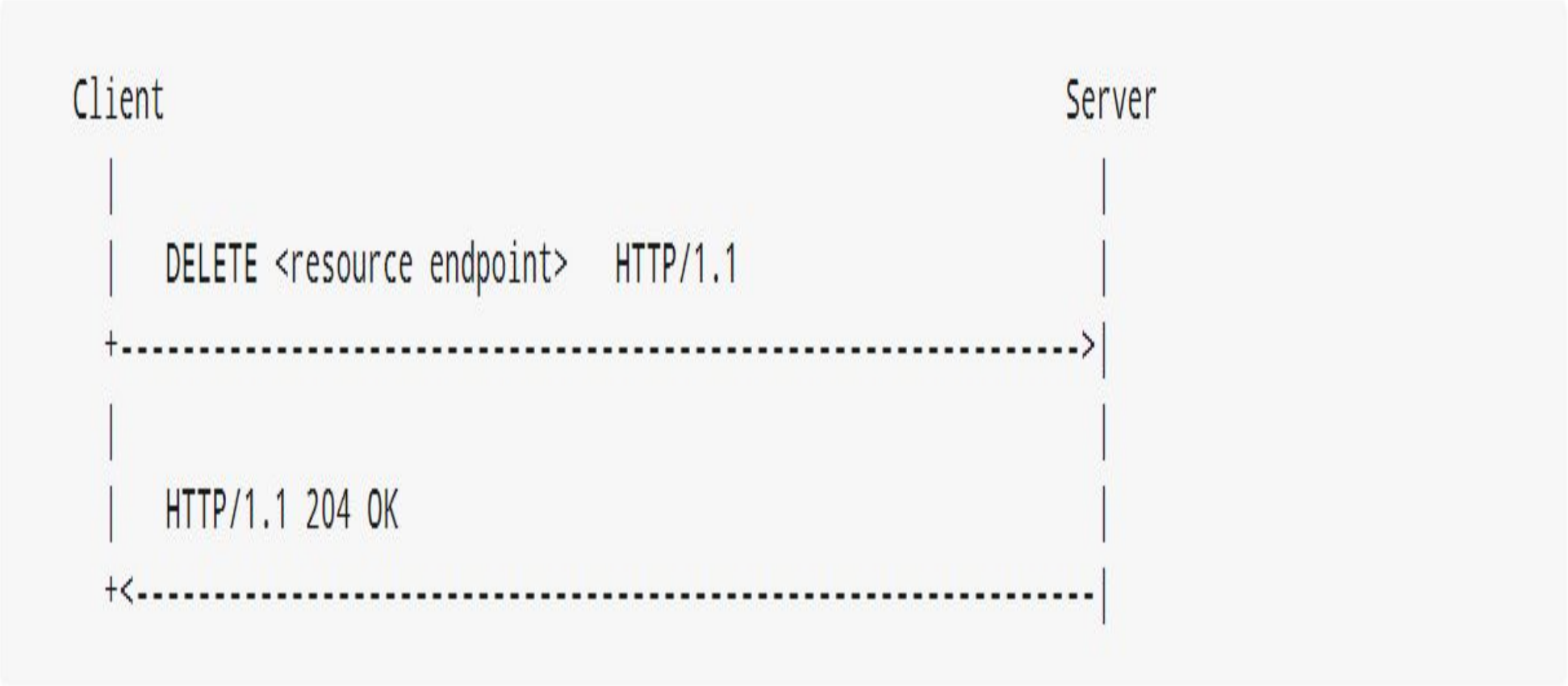
Create: Sequence diagram



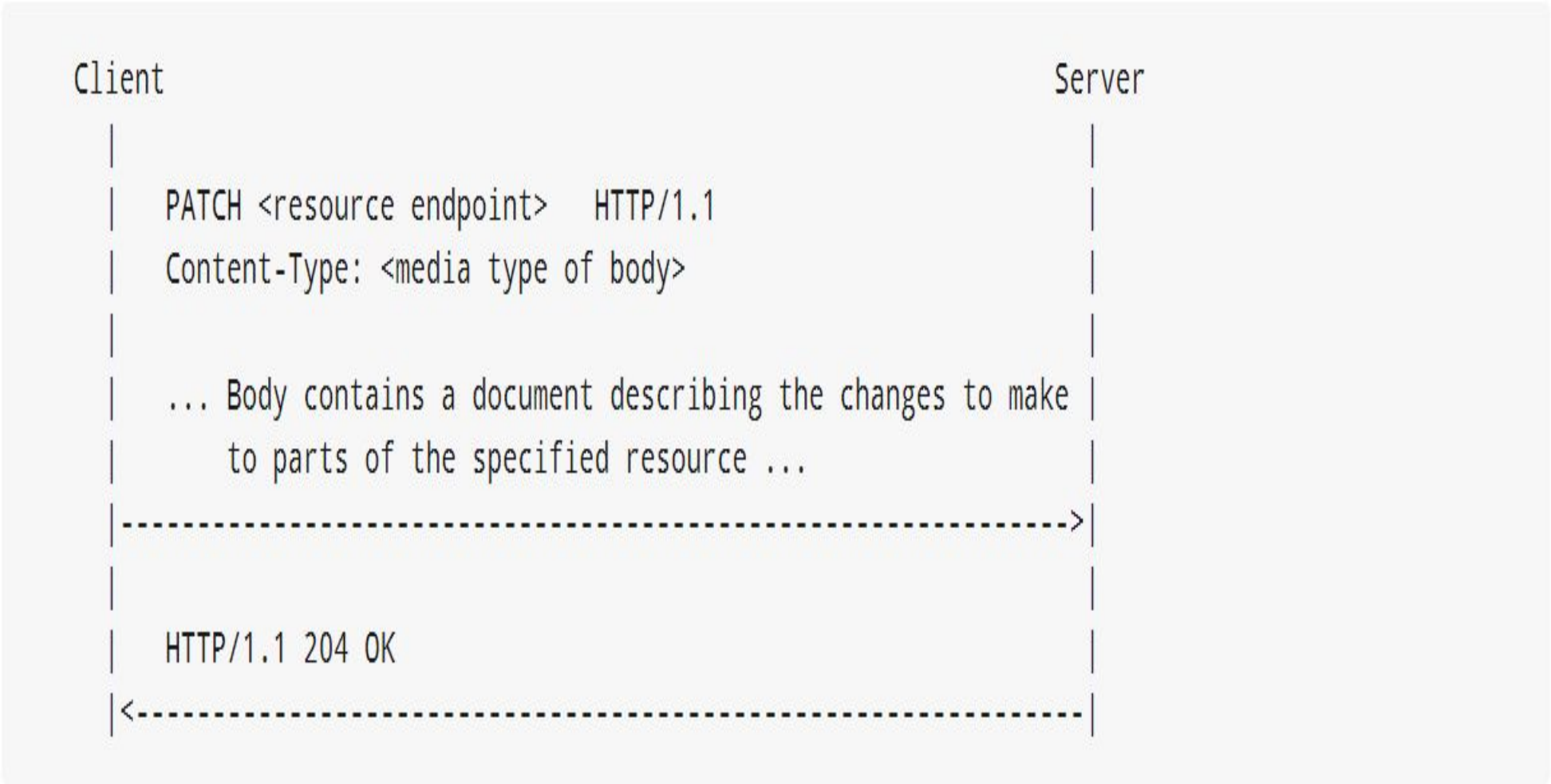
Replace : Sequence diagram



Delete : Sequence diagram



Update : Sequence diagram



Hands-on

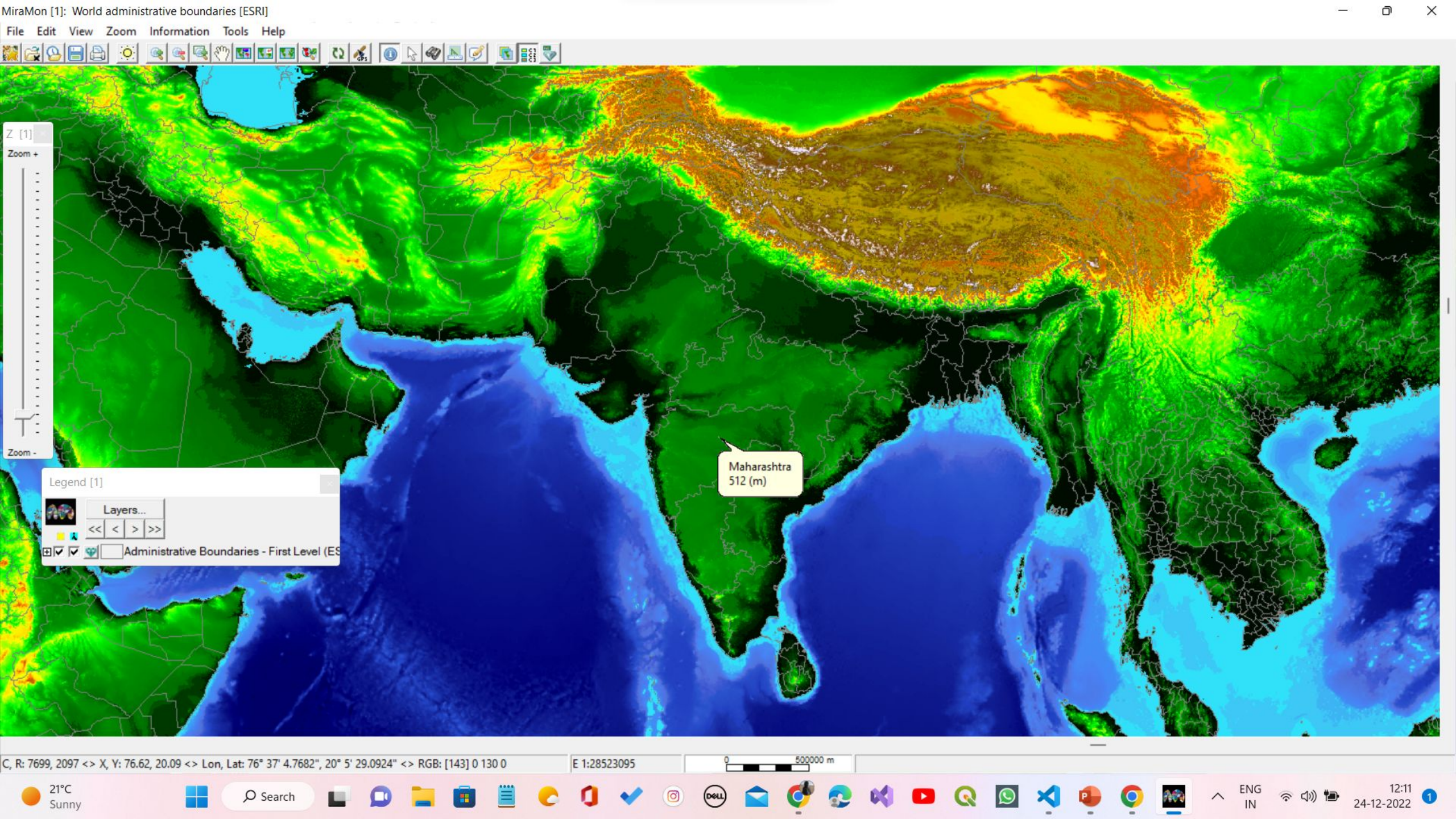
- Download and install MiraMon Map Server that implements support for multiple OGC API standard. Such as:

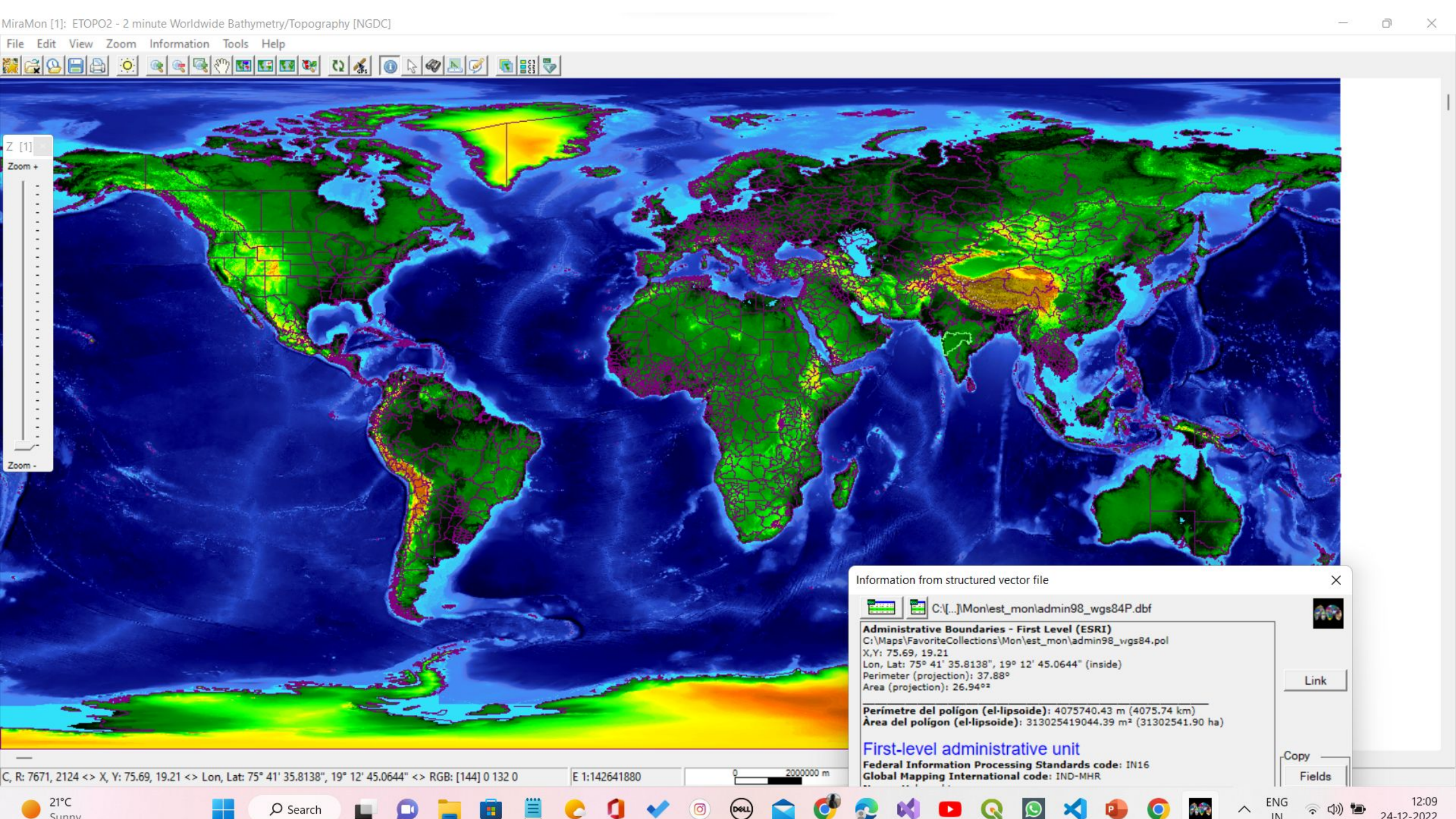
- OGC API – Map
- OGC API – Features
- OGC API - Tiles

MiraMon Map Server

- The **Centre for Ecological Research and Forestry Applications (CREAF)** at the **Autonomous University of Barcelona (UAB)** deployed an instance of the MiraMon Map Server that implements support for multiple OGC API standard.

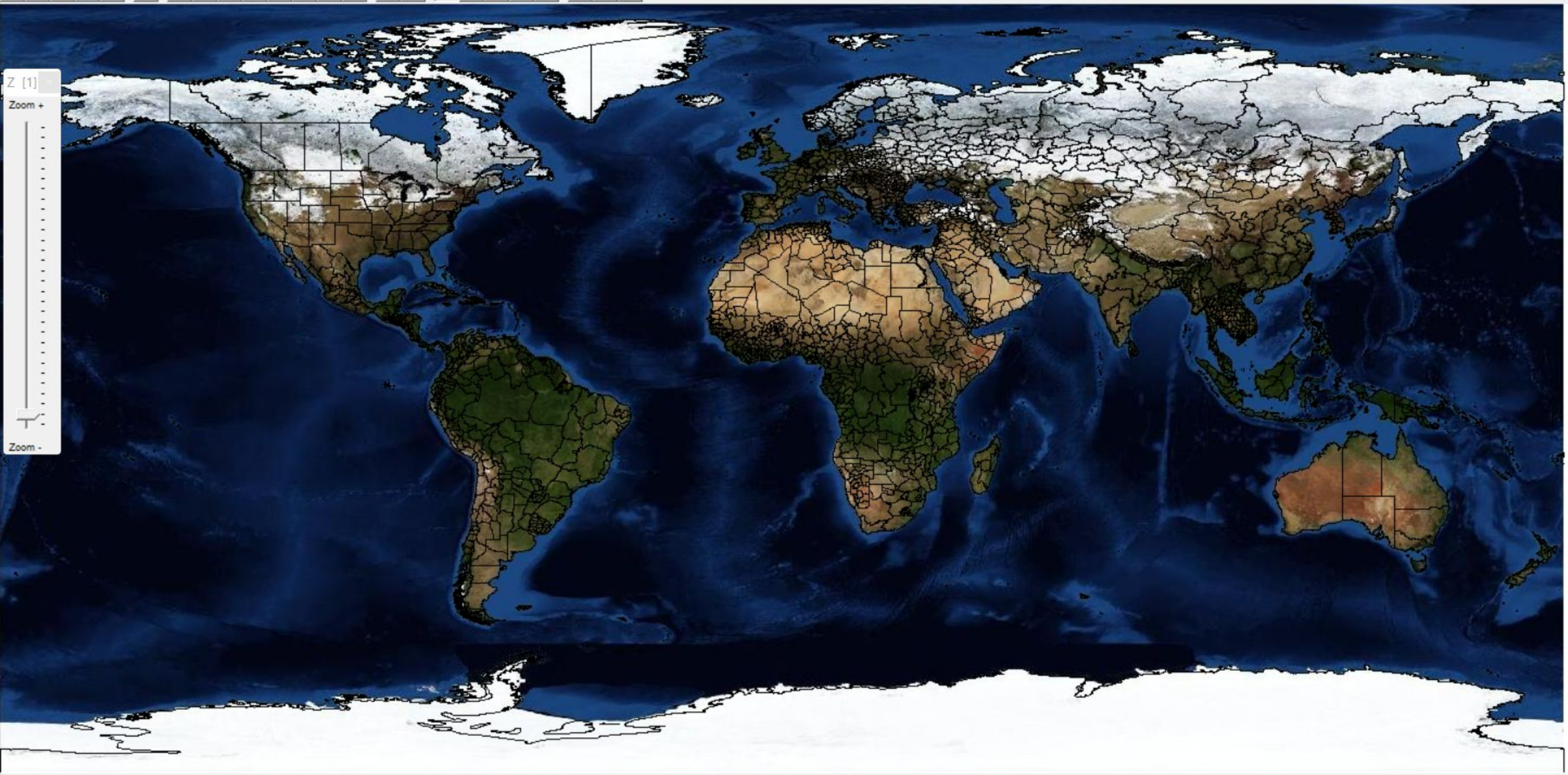
- The server is implemented as a CGI application encoded in **C language** as a part of the MiraMon suite and is interoperable with other vendors' clients **Geographic Information System (GIS) & Remote Sensing (RS)**


MiraMon [1]: World administrative boundaries [ESRI]
File
Edit
View
Zoom
Information
Tools
Help
Z [1]
Zoom +
Zoom -
Legend [1]
Layers...
<<
<
>
>>
Administrative Boundaries - First Level (ES
Maharashtra
512 (m)
C, R: 7699, 2097 <> X, Y: 76.62, 20.09 <> Lon, Lat: 76° 37' 4.7682", 20° 5' 29.0924" <> RGB: [143] 0 130 0
E 1:28523095
0
500000 m
21°C
Sunny
Search
ENG
IN
12:11
24-12-2022


MiraMon [1]: ETOPO2 - 2 minute Worldwide Bathymetry/Topography [NGDC]
File Edit View Zoom Information Tools Help
Z [1]
Zoom +
Zoom -
Information from structured vector file
C:\[...]\Mon\est_mon\admin98_wgs84P.dbf
Administrative Boundaries - First Level (ESRI)
C:\Maps\FavoriteCollections\Mon\est_mon\admin98_wgs84.pol
X,Y: 75.69, 19.21
Lon, Lat: 75° 41' 35.8138", 19° 12' 45.0644" (inside)
Perimeter (projection): 37.88°
Area (projection): 26.94°²
Perímetre del polígon (el·lipsoide): 4075740.43 m (4075.74 km)
Àrea del polígon (el·lipsoide): 313025419044.39 m² (31302541.90 ha)
First-level administrative unit
Federal Information Processing Standards code: IN16
Global Mapping International code: IND-MHR
Link
Copy
Fields
C, R: 7671, 2124 <> X, Y: 75.69, 19.21 <> Lon, Lat: 75° 41' 35.8138", 19° 12' 45.0644" <> RGB: [144] 0 132 0
E 1:142641880
0 2000000 m
21°C
Search
ENG
12:09
24-12-2022

MiraMon [1]: ETOPO2 - 2 minute Worldwide Bathymetry/Topography [NGDC]

File Edit View Zoom Information Tools Help



C, R: 502, 1901 <> X, Y: -163.27, 26.64 <> Lon, Lat: -163° 16' 0.0012", 26° 38' 30.0012" <> RGB: [13] 3 6 25

E 1:140014544

0 2000000 m

21°C Sunny

Search

ENG IN

12:10 24-12-2022

THANK YOU!

prajwalita.chavan@gmail.com
prajwalita@iitb.ac.in

#OGCAPI