OGC API workshop

None

Open Geospatial Consortium

Table of contents

1. Welcome to the OGC API workshop!

OGC is globally known for its proven widely implemented open standards. The OGC open consensus-based standards development process has evolved to move at the pace of innovation, with constant input from technology forecasting, practical prototyping, real-world testing, certification and compliance and community engagement. Today we are revolutionizing how geospatial/location information is shared, accessed, integrated, and analyzed via the OGC's revolutionary APIs- the building blocks for location information.

OGC APIs are designed to make it easy for ANYONE to provide and use geospatial data on the web, and to integrate this data with ANY other type of information. These Standards build upon the legacy of the OGC Web Service Standards (WMS, WFS, WCS, WPS, etc.), but define resource-centric APIs that take advantage of modern web development practices. This web page provides information on these Standards in a consolidated location.

These Standards are being constructed as "building blocks" that can be used to assemble novel APIs for web access to geospatial content. The building blocks are defined not only by the requirements of the Standards specified in the OGC's Standards Program, but also through interoperability prototyping and testing in the OGC's Collaborative Solutions and Innovation Program.

2. Your workshop team

Joana Simoes (OGC)

Tom Kralidis (OSGeo)

3. About this workshop

This workshop provides a combination of concepts, presentation and exercises of OGC APIs in support of discovery, access, visualization and processing in support of FAIR data principles.

Each deep dive module starts with an "Audience" and "Learning Objectives" sections:

Abstract

Describes what this module is about.

Learning Objectives

Describes what students should be able to do, after the completion of the module.

Exercises are indicated as follows:

Example exercise

A section marked like this indicates that you can try out the exercise.

Also you will notice notes sections within the text:

Note

Highlights some important aspect.

Examples are indicated as follows:

Bash shell

```
curl -X 'GET' 'https://myrequest.com'
```

JSON

```
{
    "title": "my cool collection title",
    "description": "my cool collection description"
}
```

4. Workshop location and materials

This workshop is always provided live at ogcapi-workshop.ogc.org.

The workshop contents, wiki and issue tracker are managed on GitHub at github.com/opengeospatial/ogcapi-workshop.

4.1 Printing this workshop

To print this workshop, navigate to the print page and select *File* > *Print* > *Save as PDF*.

5. Support

For issues/bugs/suggestions or improvements/contributions, please use the GitHub issue tracker.

Contributions are always encouraged and welcome!

6. Cite this workshop

DOI 10.5281/zenodo.15846453

6.1 Now, on to the workshop. Ready? Let's go!

7. Background and context

The geospatial domain has a long history of efforts focused on data discovery, access, and visualization.

This page provides background and history of geospatial web services, web mapping, and distributed computing platforms.

7.1 Geospatial API evolution

The 1990s saw the initial implementation of Service-Oriented Architecture (SOA). The first OGC Web Map Service (WMS) standard was published in 1999, providing a vendor-neutral approach to visualizing maps of geospatial data on a web page. Web services had strong roots in XML-RPC (eXtensible Markup Language over Remote Procedure Call) and CORBA, and standards and technologies such as SOAP, WSDL and UDDI began to emerge as as means to describe, discover, and perform request/response workflow via a web interface.

The 2000s saw continued development of OGC Web Service standards such as Web Feature Service (WFS), Catalogue Service for the Web (CSW), Web Coverage Service (WCS), Web Processing Service (WPS), and others. In addition, Geography Markup Language (GML) became an official OGC standard in support of standardized (vector) data exchange over the Internet. Web services were typically designed with the concept of a relational database model (RDBMS) as the backend data/metadata repository.

In the mid 2000s, JavaScript, AJAX and slippy maps/tiles began to emerge in the web mapping domain, which provided Web 2.0 design patterns which resulted in more responsive maps on web pages over the Web.

7.2 Realities of web service architectures

OGC Web service standards provided state of the art approaches for data discovery, access, and visualization. At the same time, as the concept of Web architecture evolved, various characteristics of Web service standards showed room for improvement in order to evolve:

- XML: while XML was/is a proven and extensible encoding/representation, working with XML in a web development environment proved cumbersome (parsing into dedicated objects). As well, the verbose nature of XML proved expensive for working over the Web for transferring large payloads of data
- Building a specialized transport layer on top of HTTP: HTTP (the protocol) itself provided native request/response mechanisms (error codes, content negotiation), which were typically additional defined within Web service standards. For example, an HTTP 400 exception natively communicates a faulty request, whereas an HTTP 200 exception indicates a successful response. Web service standards were commonly developed where a 200 response could indicate a faulty request (where the client would have to inspect the content of an HTTP response, instead of the actual HTTP status code
- Specifications: Web service specifications were typically feature complete and developed with a 100% solution in mind, which proved challenging to implement a fully complient server or client, for example
- Web challenges: Web service specifications were difficult to implement for web developers and typically required specialized domain expertise in geospatial to interpret and understand specification requirements. In addition, Web services were difficult to integrate with mainstream Internet search engines

7.3 Being webby: a new paradigm

In 2017, the W3C published the Spatial Data on the Web Best Practices, which provided recommendations on data formats, identifiers, access, licensing, and provenance. The goal of this best practice was to provide a baseline of recommendations to integrate geospatial data and services with mainline Web practices and design patterns. In addition, the OGC published the OGC API Whitepaper describing, discussing APIs and next steps for the OGC at the time. It became obvious that a "clean break" was needed, in order for OGC APIs to become more "of the Web" and lower barrier for non-domain experts.

A new paradigm emerged, which highlighted the following concepts:

- being webby (humans, search engines)
- developer friendly
- lightweight specification development
- moving from service oriented to resource oriented: removing HTTP use as a tunnel:
- service oriented: `/ows?request=GetFeature&typename=roads&featureid=5`
- resource oriented: `/api/collections/roads/items/5`
- modular specification development
- core and extension specifications, allowing for low barrier implementation and adoption for mass market/the Web

In 2018, the OGC began to ran various hackathons, starting work on WFS3 (now OGC API - Features - Part 1: Core), as well as a Weather on the Web API (now OGC API - Environmental Data Retrieval). This represented the origins of the OGC API movement!

The development and working processes of OGC standards development themselves evolved during this time. OGC API specifications began to be developed on public GitHub repositories, allowing for anyone in the public to discuss and collaboration for a given OGC API standard in the open on GitHub. In addition, the specifications themselves began to be developed in AsciiDoc (an open format for documentation/markup), and made available as HTML pages (and PDF). Collaborative tools used by OGC also proliferated, such as Gitter/Element as well as Discord.

Note

OGC standards, while primarily developed on GitHub, are voted on by the OGC membership.

7.4 Summary

Geospatial Web Services and APIs have been a long running activity in the geospatial domain. OGC API standards are designed on the lessons learned of past efforts, and built to be low barrier, with a focus on resources (data/content!) using modern web development principles and practices.

8. Overview and main concepts

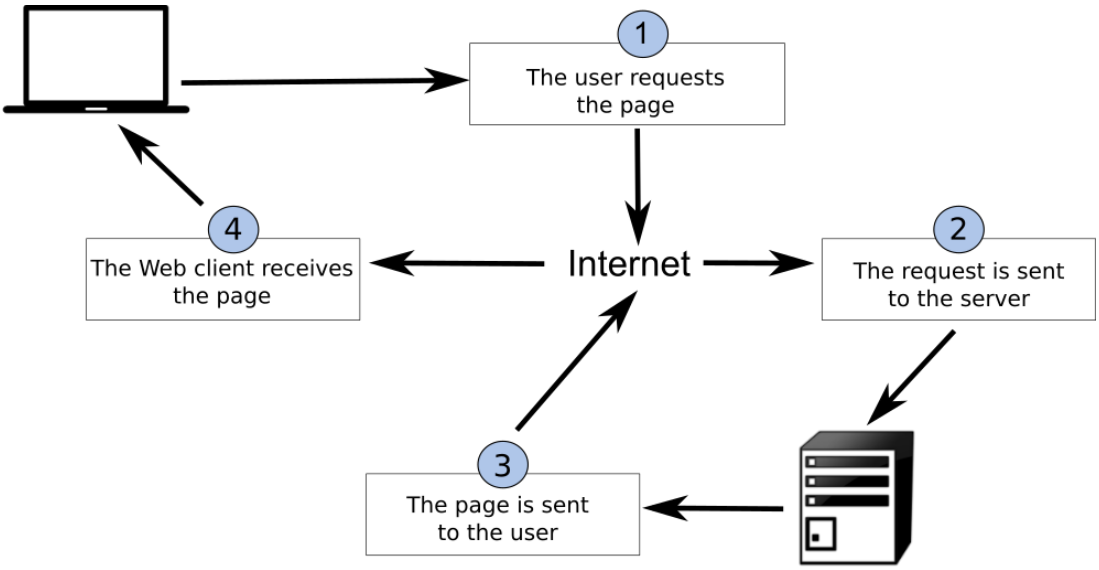
The core of Web APIs can be summarized as:

- interfaces: the way in which "conversations" happen between APIs and clients of them
- encodings: the "formats" of contents provided by an API

8.1 Client / server

In a typical client / server environment, a client is asking a server to perform an action (for example, requesting data), with the ability to add additional instructions such as querying, filtering and what format for the API to provide as part of the response.

The image below, taken from Introduction to GIS illustrates the concept of the request / response lifecycle between a client and a server.



8.2 Web architecture

8.2.1 REST

REpresentational State Transfer (REST) is an architectural style for the Web. The core concepts of REST are:

- HTTP verbs (GET/PUT/POST/DELETE)
- HTTP codes (200, 201, 404, etc.)
- URIs to identify resources
- Content negotiation (media types)
- Stateless

Implementing REST results in a simpler, low barrier architecture that is based on web primitives. This enables systems and applications to focus more on domain/business requirements.

Know your HTTP!

- verbs: https://http.dev/methods
- status codes: https://http.dev/status

8.2.2 JSON

JSON (JavaScript Object Notation) is a compact and very easy to understand encoding, which is very popular among web developers. JSON is the primary encoding used in RESTful web services and APIs, and is by nature extensible.

Let's compare JSON and XML in a simple example:

An example XML document (75 bytes):

```
<order>
    <orderID>123</orderId>
    <status>completed</status>
</order>
```

The same document as JSON (46 bytes):

```
{
  "orderID": 123,
  "status": "completed"
}
```

Here, we see a more compact representation using JSON. In addition, it is easier to determine the underlying data type literals (integers, strings, etc.) by parsing the document itself.

Note

JSON Schema is the JSON equivalent to W3C XML Schema, providing a language to define the content model of a JSON document. A JSON document can choose to implement a JSON Schema, or not, depending on a given application's requirements for data validation and integrity

8.3 OGC APIs

This section provides a high level overview of OGC API standards support.

Cite

The OGC API family of standards are being developed to make it easy for anyone to provide geospatial data to the web. These standards build upon the legacy of the OGC Web Service standards (WMS, WFS, WCS, WPS, etc.), but define resource-centric APIs that take advantage of modern web development practices. This web page provides information on these standards in a consolidated location.

These standards are being constructed as "building blocks" that can be used to assemble novel APIs for web access to geospatial content. The building blocks are defined not only by the requirements of the specific standards, but also through interoperability prototyping and testing in OGC's Innovation Program.

8.3.1 OGC API - Common

OGC API - Common is a common framework used in all OGC API's. OGC API - Common provides the following functionality:

- based on OpenAPI 3.0
- HTML and JSON as the dominant encodings, alternative encodings are possible
- common and shared endpoints such as:
- `/` (landing page)
- `/conformance`
- `/openapi`
- `/collections`
- `/collections/foo`
- common aspects such as pagination, links between resources, basic filtering, query parameters (`bbox`, `datetime`, etc.)

OGC API - Common allows for specification developers to focus on the key functionality of a given API (i.e. data access, etc.) while using common constructs. This harmonizes OGC API standards and enables deeper integration with less code. This also allows for OGC API client software to be more streamlined.

For more details about this standard, please refer to the OGC API - Common section.

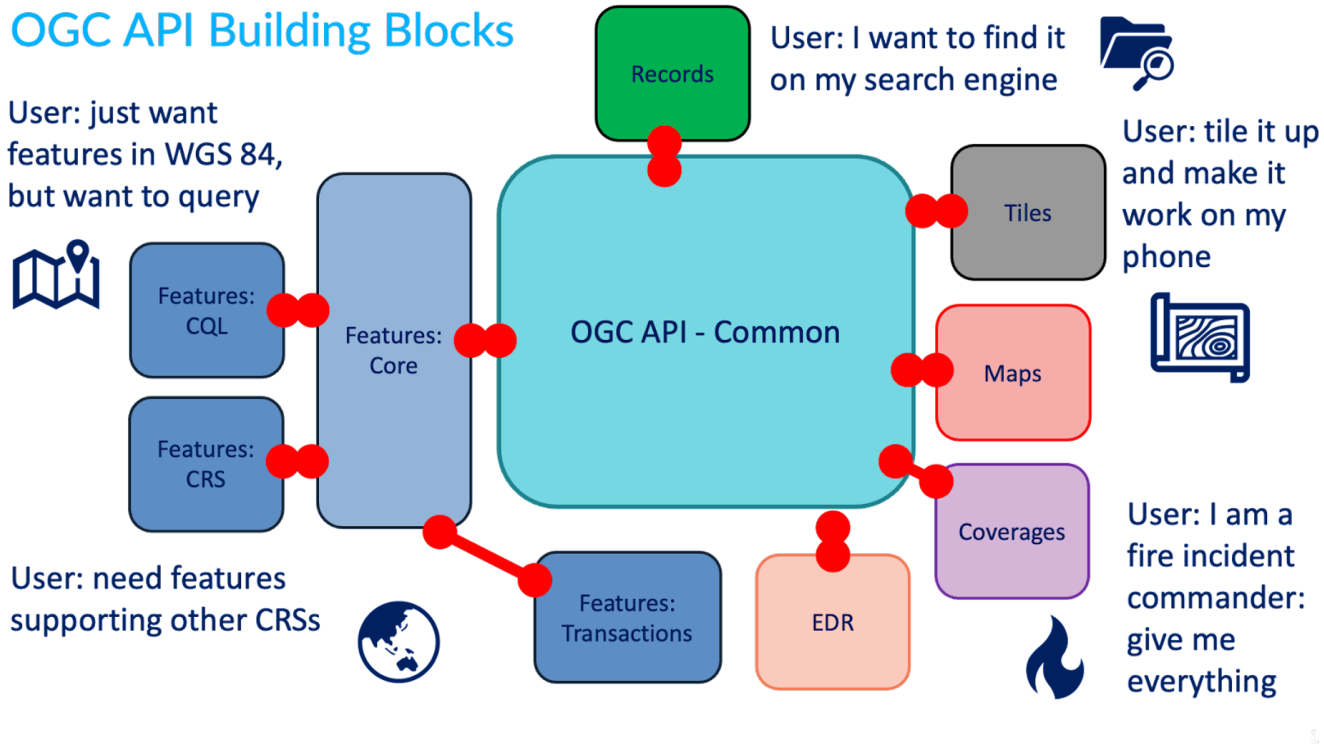
8.3.2 Approved Standards

The below OGC API standards have been approved and available for use. Note that these standards have 1 or more "Parts" or extensions that enable specific functionality. "Part 1" of a given standard represents the most basic capabilities. Additional parts can also be implemented as building blocks.

- OGC API - Features offers the capability to create, modify, and query spatial data on the Web and specifies requirements and recommendations for APIs that want to follow a standard way of sharing feature data
- OGC API - Environmental Data Retrieval provides a family of lightweight interfaces to access Environmental Data resources. Each resource addressed by an EDR API maps to a defined query pattern
- OGC API - Maps offers a modern approach to the OGC Web Map Service (WMS) standard for provision map and raster content
- OGC API - Processes allows for processing tools to be called and combined from many sources and applied to data in other OGC API resources though a simple API
- OGC API - Tiles provides extended functionality to other OGC API Standards to deliver vector tiles, map tiles, and other tiled data
- Moving Features defines an API that provides access to data representing features that move as rigid bodies
- OGC API - Records provides discovery and access to metadata about geo spatial resources.

8.3.3 OGC API building blocks

The OGC API approach allows for modularity and "profiling" of APIs depending on your requirements. This means you can mix and match OGC APIs together.

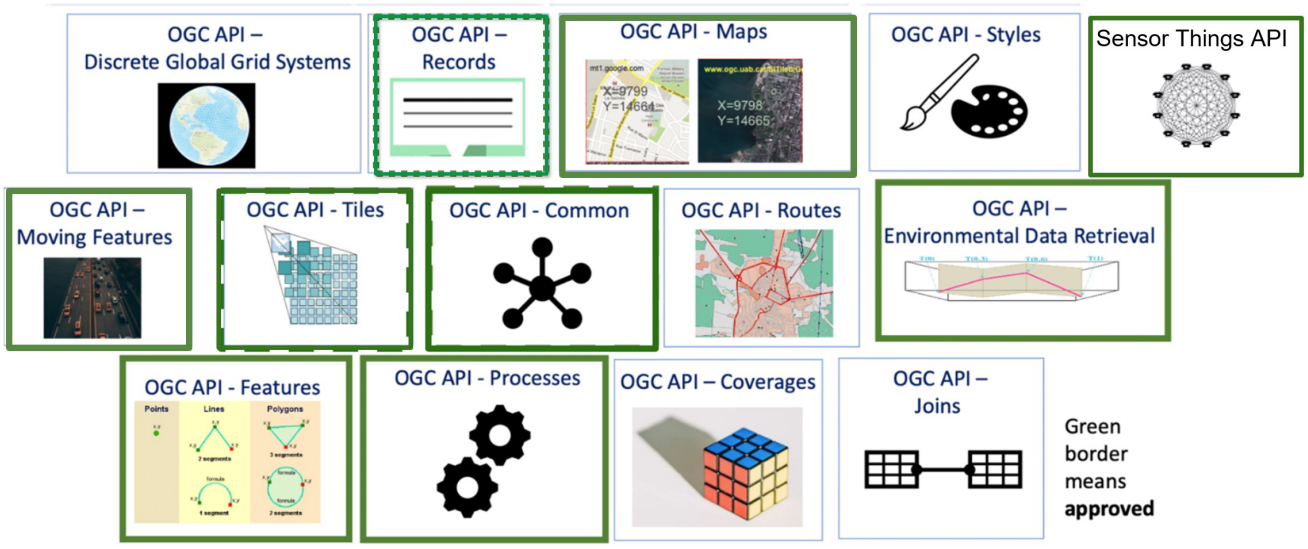


You can read more about this topic in the building blocks website.

8.3.4 In development

The OGC API effort is rapidly evolving. Numerous OGC API standards are in development:

- Routes provides access to routing data
- Styles defines a Web API that enables map servers, clients as well as visual style editors, to manage and fetch styles
- 3D GeoVolumes facilitates efficient discovery of and access to 3D content in multiple formats based on a space-centric perspective
- Joins supports the joining of data, from multiple sources, with feature collections or directly with other input files
- Discrete Global Grid System enables applications to organise and access data arranged according to a Discrete Global Grid System (DGGS)
- Connected Systems is intended to act as a bridge between static data (geographic and other domain features) and dynamic data (observations of these feature properties, and commands/actuations that change these feature properties)



8.3.5 OpenAPI

Core to OGC API - Common is the OpenAPI initiative to help describe and document an API. OpenAPI defines its structure in an OpenAPI document. OGC API - Common suggests this document to be located at `/openapi`. For example, with pygeoapi in a browser this URL opens an interactive HTML page which facilitates an API query. Append `?f=json` to view the document in JSON. The OpenAPI document indicates which endpoints are available in the service, which parameters it accepts and what types of responses can be expected. The OpenAPI document is a similar concept to Capabilities XML as part of the first generation OGC Web Service standards.

OpenAPI Specification parsing in a browser

A common approach to interact with Open API's using json is to use a program like Postman. Also there are browser plugins which enable to define api requests interactively within a browser. For firefox download the plugin poster. For Chrome and Edge use Boomerang. In Boomerang you can create individual web requests, but also load the open api specification document and interact with any of the advertised endpoints.

The OpenAPI community provides various tools, such as a validator for OAS documents or generate code as a starting point for client development.

8.4 Content and format standards

OGC APIs are typically format agnostic for data. This means an OGC API can provide any format of data or metadata (JSON, YAML, XML, HTML, etc.).

JSON is a core format that is machine readable and easy to parse and handle by client software and tools. JSON is easily decoded/encoded into native objects in numerous programming languages (Python dictionaries, JavaScript objects, etc.). OGC API - Common provides uniform JSON formats for the various endpoints it supports.

Specific OGC API standards may specify domain specific formats (for example, GeoJSON for OGC API - Features, GeoTIFF for OGC API - Coverages, ISO 19115/19139 for OGC API - Records, etc.), depending on the data or metadata type(s).

8.5 Summary

OGC APIs leverage core principles of Web architecture, providing support for discovery, access, visualization, processing of geospatial data, in alignment with industry standards for maximum interoperability on the Web.

9. Compliance and certification

The OGC Compliance Program provides the resources, procedures, and policies to certify products for compliance with one or more OGC standards.

The primary purpose of the program is to increase systems interoperability while reducing technology risks by providing a process whereby compliance with OGC standards can be tested.

You can browse the OGC compliance product page to check the list of products certified by OGC.

9.1 TEAM Engine

In order to be certified, a product needs to passed the tests on the OGC validator. The OGC validation infrastructure is based on TEAM Engine, a Free and Open Source tool which is being incubated as an OSGeo project.

Developers can either use the hosted version of TEAM Engine on CITE, or install and integrate it locally into their own pipeline.

OGC API test suites

We will refer to the available test suites in the context of each standard, during the OGC API Deep Dive.

9.2 Reference implementations

A reference implementation (RI) is a fully functional, licensed copy of a tested, branded software that has passed the test for an associated conformance class in a version of an Implementation Standard and that is free and publicly available for testing via a web service or download.

- RI do not need to comply to all the conformance classes in the standard, but they should conform at least to the core.
- RI should be available, on a reliable server that has a high treshold for uptime.
- OGC will provide an incentive to the first two RI that pass the test related to a conformance class within a version of an Implementation Standard

More information about RI is available on Compliance Testing Program Policies & Procedures.

Compliance badge	Compliance badge	Compliance badge

9.3 Summary

OGC compliance and certification provides an organization a level of confidence that a product has been deemed compatible for maximum interoperability against a given OGC specification.

10. API deep dive

10.1 Primer

The deep dives will focus on numerous OGC API standards, starting from the foundational OGC API - Common, followed by API implementations for various geospatial data types and workflows. Discovery, access, visualization and processing are core workflows in the geospatial domain. The deep dives are designed for you to "dive deep" into to better understand their purpose and applicability to your requirements/interests.

Each API deep dive will consist of the following components:

- Introduction: an overview of the API, its core functionality, and how it fits into the greater OGC API ecosystem
- Resources: a detailed description of each URL / endpoint supported by the API

> **Note**
>
> "Resource" can be an overloaded term. The "Resources" section in each deep dive describe HTTP methods, URL paths/endpoints and explanations of functionality.

10.2 Ready to deep dive into the OGC APIs? Let's go!

10.3 OGC API - Common

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Common draft standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Common standard is
- Describe what can be done with OGC API - Common as a building block

10.3.1 Introduction

OGC API - Common specifies building blocks that are shared by most or all OGC API Standards to ensure consistency across the family. In the course of developing Resource Oriented Architectures and Web APIs, some practices proved to be common accross all OGC API standards. The purpose of this standard is to document those practices. It also serves as a **common foundation** upon which all OGC APIs will be built.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Common - Part 1: Core** standard or **OGC API - Common - Part 2: Geospatial Data** draft standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Common - Part 1: Core** standard and **OGC API - Common - Part 2: Geospatial Data** draft standard for additional detail.

Background

History

OGC API Common standard serves as the "OWS Common" standard for OGC Resource Oriented APIs. The OGC API - Common SWG charter was created in 2020 OGC API - Common - Part 1: Core was approved on February 2023.

Versions

OGC API - Common - Part 1: Core version 1.0.0 is the current latest version

USAGE

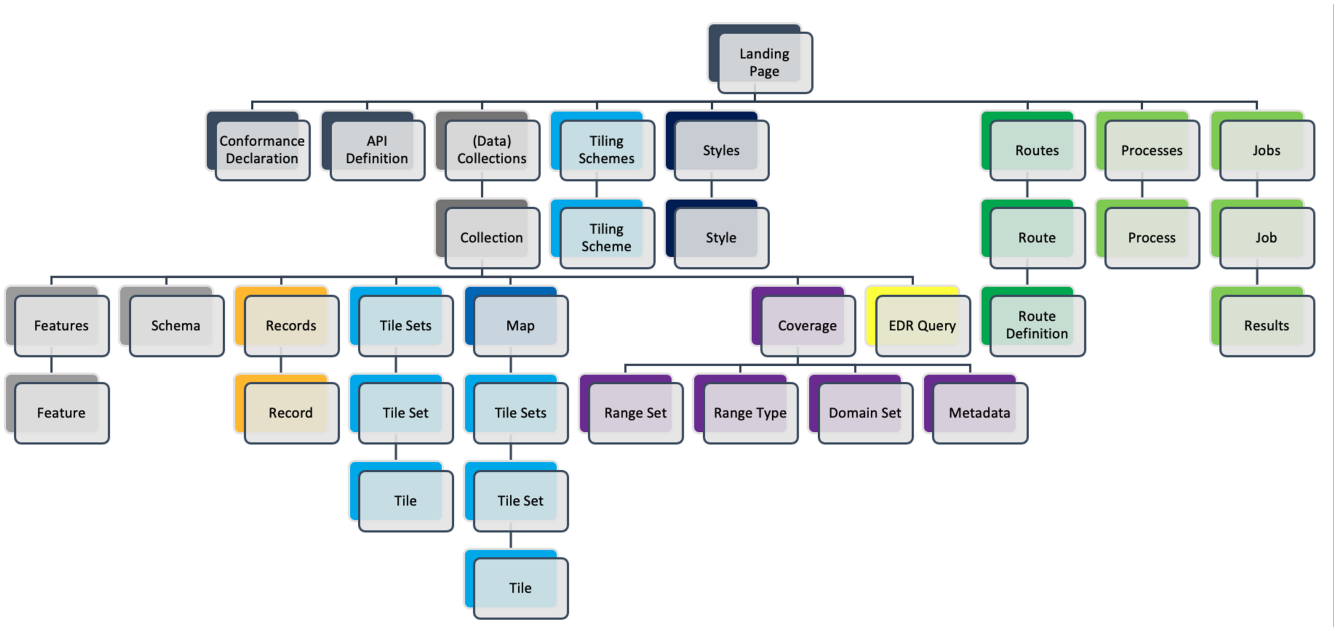
This specification identifies resources, captures compliance classes, and specifies requirements which are applicable to all OGC API standards. It should be included as a normative reference by all such standards.

- The **OGC API - Common - Part 1: Core** Standard defines the resources and operations which SHALL be common to all OGC API standards. This Standard defines the minimal requirements for an API to be discovered and used by any client.
- The Draft **OGC API - Common - Part 2: Geospatial Data** Standard provides a common connection between the API landing page and resource-specific details. That connection includes metadata which describes the collections of hosted resources, common parameters for selecting subsets of those collections, and URI templates for identifying the above.

In addition, OGC API - Common provides some non-normative information through the OGC API-Common Users Guide.

RELATION TO OTHER STANDARDS

The image below shows the resource architecture in OGC API. OGC API - Common provides a common foundation to all OGC APIs.



Overview of Resources

OGC API - Common - Part 1: Core defines the resources listed in the following table:

Resource	Method	Path	Purpose
Landing page	GET	/	Retrieves the landing page. The purpose of the landing page is to provide clients with a starting point for using the API. Any resource exposed through an API can be accessed by following paths or links starting from the landing page. The landing page includes three metadata elements; title, description, and attribution. Only the title is required. These three elements describe the API as a whole. Clients can expect to encounter metadata which is more resource-specific as they follow links and paths from the landing page.
Conformance declaration	GET	/conformance	Provides a list declaring the modules that are implemented by the API. These modules are referred to as Conformance Classes. The list of Conformance Classes is key to understanding and using an OGC Web API.
API definition	GET	/api	Retrieves the API definition which describes the capabilities provided by that API. This resource can be used by developers to understand the API, by software clients to connect to the server, and by development tools to support the implementation of servers and clients. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.

The purpose of the draft **OGC API - Common - Part 2: Geospatial Data** Standard is to provide a common connection between the API landing page and resource-specific details. The table below defines the resources listed in this part.

Resource	Method	Path	Purpose
Collections	GET	/collections	Retrieves information which describes the set of supported Collections.
Collection	GET	/collections/ {collectionId}	Retrieves descriptive information about a specific Collection.

Providing a **common foundation**, OGC API - Common is meant to be implemented by "downstream" OGC API standards in a uniform and consistent manner. Examples of OGC API - Common resources will be shown in the context of other OGC API standards.

10.3.2 Summary

OGC API - Common documents the set of common practices and shared requirements that have emerged from the development of Resource Oriented Architectures and Web APIs within the OGC. The standard defines resources and access mechanisms which are useful for a client seeking to understand the offerings and capabilities of an API, as well as a connection between the API landing page and resource-specific details. In this deep dive we provided an overview of the standard and look at the resources on part 1 and part 2 (draft).

10.4 OGC API - Features

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Features standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Features standard is
- Describe what can be done with OGC API - Features implementations
- Understand the main resources offered by OGC API - Features implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Features implementation
- Understand how to issue requests to an implementation of OGC API - Features
- Be able to find an OGC API - Features endpoint and use it through a client

10.4.1 Introduction

OGC API - Features is a multi-part standard that offers the capability to create, modify, and query spatial data on the Web and specifies requirements and recommendations for APIs that want to follow a standard way of sharing feature data. **OGC API - Features - Part 1: Core** describes the mandatory capabilities that every implementing service must support and is restricted to read-access to spatial data. Additional capabilities like support for different CRS, richer queries and creating and modifying data are specified in additional parts.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Features - Part 1: Core** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Features - Part 1: Core** standard for additional detail.

Background

History

While in draft form and prior to February 2019, **OGC API - Features - Part 1: Core** was referred to as WFS3.0.

Versions

OGC API - Features - Part 1: Core version 1.0.1, **OGC API - Features - Part 2: Coordinate Reference Systems by Reference** version 1.0.1 and **OGC API - Features - Part 3: Filtering** version 1.0.1 are the current latest versions

Test suite

Test suites are available for:

- OGC API - Features - Part 1
- OGC API - Features - Part 2

All of the test suites are available from the OGC Validator.

Implementations

Implementations can be found on the implementations page.

USAGE

The standard provides a standard interface for requesting vector geospatial data consisting of geographic features and their properties. The benefit of this is that client applications can request source data from multiple implementations of the API, and then render the data for display or process the data further as part of a workflow. The standard enables the data to be accessed consistently with other data. Feature properties encoded using common data types such as text strings, date and time can also be accessed consistently.

- **OGC API - Features - Part 1: Core** specifies discovery and query operations that are implemented using the HTTP GET method. Support for additional methods (in particular POST, PUT, DELETE, PATCH) will be specified in additional parts. Government agencies, private organisations and academic institutes use this standard to publish vector geospatial datasets in a way that makes it easier for receiving organisations to compile new maps or conduct analysis on the supplied data. In Part 1 the default spatial Coordinate Reference System (CRS) is WGS 84 longitude/latitude with or without height.
- **OGC API - Features - Part 2: Coordinate Reference Systems By Reference** extends Part 1 to support presenting geometry-valued properties in a response document in additional CRSs. Each supported CRS must be identified by a URI such as: `http://www.opengis.net/def/crs/EPSG/0/4326`.
- **OGC API - Features - Part 3: Filtering** defines query parameters (`filter`, `filter-lang`, `filter-crs`) to specify filter criteria in a request to an API and the `Queryables` resource that declares the properties of data in a collection that can be used in filter expressions.

In addition to the approved parts above, The OGC API - Features Standards Working Group (SWG) is working on the following drafts:

- *Draft* **OGC API - Features - Part 4: Create, Replace, Update and Delete** defines the behaviour of an API that allows resource instances to be added, replaced, modified and/or removed for a collection.
- *Draft* **OGC API - Features - Part 5: Schemas** specifies how features can be described by a logical schema and how such schemas are published in an OGC Web API implementation.
- *Draft* **OGC API - Features - Part 6: Property Selection** specifies how the representation of a resource can be reduced to selected properties of the resource using a query parameter.
- *Draft* **OGC API - Features - Part 7: Geometry Simplification** specifies how the representation of geometry can be simplified using a query parameter.
- *Draft* **OGC API - Features - Part 8: Sorting** defines query parameters (sortby) to specify sorting criteria in a request to an API and the Sortables resource that declares the properties of data in a collection that can be used in sort by expressions.
- *Draft* **OGC API - Features - Part 9: Text Search** adds a query parameter to the OGC API Features suite of standards to support text or keyword searches on text fields.
- *Draft* **OGC API - Features - Part 10: Search/Queries** adds support to dynamically fetch features from multiple collections at a time.

> **Note**
>
> The rest of this tutorial will focus on the core part of the standard.

RELATION TO OTHER OGC STANDARDS

- OGC Web Feature Service Interface Standard (WFS): The WFS standard is more appropriate when working with client applications that only support classic OGC Web Services. Note as well that WFS adopts the Geography Markup Language (GML) as a default data format. In contrast, OGC API - Features includes recommendations to support HTML and GeoJSON as encodings, where practical. Implementations of OGC API - Features may also optionally support GML, as well as other vector formats.

Overview of Resources

OGC API - Features - Part 1: Core defines the resources listed in the following table.

Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
API definition	GET	/api	This resource provides metadata about the API itself. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.
Feature collections	GET	/collections	This resource lists the feature collections that are offered through the API.
Feature collection	GET	/collections/{collectionId}	This resource describes the feature collection identified in the path.
Features	GET	/collections/{collectionId}/items	This resource presents the features that are contained in the collection.
Feature	GET	/collections/{collectionId}/items/{featureId}	This resource presents the feature that is identified in the path

Example

This demonstration server publishes vector geospatial data through an interface that conforms to OGC API - Features.

An example request that can be used to retrieve data from the Portuguese Points of Interest feature collection is https://demo.pygeoapi.io/master/collections/ogr_gpkg_poi/items?f=html

Note that the response to the request is HTML in this case.

Alternatively, the same data can be retrieved in GeoJSON format, through the request https://demo.pygeoapi.io/master/collections/ogr_gpkg_poi/items?f=json

A client application can then retrieve the GeoJSON document and display or process it.

10.4.2 Resources

This section provides basic information about the types of resources that OGC API - Features offers.

Each resource provides **links** to related resources. This enables a client application to navigate the resources, from the landing page through to the individual features. The server identifies the relationship between a resource and other linked resources through a **link relation type**, represented by the attribute `rel`. The link relation types used by implementations of the **OGC API - Features - Part 1: Core** can be found in Section 5.2 of the standard.

Landing page

The landing page is the top-level resource that serves as an entry point. A client application needs to know the location of the landing page of the server. From the landing page, the client application can then retrieve links to the Conformance declaration, Collection and API definition paths. An example landing page is at https://demo.ldproxy.net/daraa?f=json

The link to the API definition is identified through the `service-desc` and `service-doc` link relation types.

The link to the Conformance declaration is identified through the `conformance` link relation type.

The link to the Collections is identified through the `data` link relation type.

An extract from the landing page of a demo server is shown below.


```
{
  "title": "Daraa",
  "description": "This is a test dataset used in the Open Portrayal Framework thread in the OGC Testbed-15 as well as the OGC Vector Tiles Pilot Phase 2. The
data is based on OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.",
  "attribution": "US National Geospatial Intelligence Agency (NGA)",
  "links": [
    {
      "rel": "self",
      "type": "application/json",
      "title": "This document",
      "href": "https://demo.ldproxy.net/daraa?f=json"
    },
    {
      "rel": "service-desc",
      "type": "application/vnd.oai.openapi+json;version=3.0",
      "title": "Definition of the API in OpenAPI 3.0",
      "href": "https://demo.ldproxy.net/daraa/api?f=json"
    },
    {
      "rel": "conformance",
      "title": "OGC API conformance classes implemented by this server",
      "href": "https://demo.ldproxy.net/daraa/conformance"
    },
    {
      "rel": "data",
      "title": "Access the data",
      "href": "https://demo.ldproxy.net/daraa/collections"
    }
  ]
}
```


Conformance declarations

An implementation of OGC API - Features describes the capabilities that it supports by declaring which conformance classes it implements. The Conformance declaration states the conformance classes from standards or community specifications, identified by a URI, that the API conforms to. Clients can then use this information, although they are not required to. Accessing the Conformance declaration using HTTP GET returns the list of URIs of conformance classes implemented by the server. Conformance classes describe the behavior a server should implement in order to meet one or more sets of requirements specified in a standard.

Below is an extract from the response to the request https://demo.ldproxy.net/daraa/conformance?f=json

Notice that the example shows a link relation type called `alternate` which identifies a way to retrieve an alternative representation of the information provided by the resource. In this case the `alternate` link relation is referencing an HTML representation of the conformance declaration.


```
{
  "links": [
    {
      "rel": "alternate",
      "type": "text/html",
      "title": "This document as HTML",
      "href": "https://demo.ldproxy.net/daraa/conformance?f=html"
    },
    {
      "rel": "self",
      "type": "application/json",
      "title": "This document",
      "href": "https://demo.ldproxy.net/daraa/conformance?f=json"
    }
  ]
"conformsTo" : ["http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core", "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson", "http://
www.opengis.net/spec/ogcapi-features-1/1.0/conf/html", "http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30", "http://www.opengis.net/spec/ogcapi-
features-2/1.0/conf/crs", "http://www.opengis.net/spec/ogcapi-features-3/0.0/conf/features-filter", "http://www.opengis.net/spec/ogcapi-features-3/0.0/conf/
filter", "http://www.opengis.net/spec/ogcapi-features-3/0.0/conf/queryables", "http://www.opengis.net/spec/ogcapi-features-3/0.0/conf/queryables-query-
parameters"]
}
```


API Definition

The API definition describes the capabilities of the server. It can be used by developers to understand the API, by software clients to connect to the server, or by development tools to support the implementation of servers and clients. Accessing the API definition using HTTP GET returns a description of the API.

There are conformance classes to provide the API definition using Open API. Some servers also return a human-readable representation of the definition in HTML, using tools such as Redoc or Swagger.

This is an extract of an API definition, which uses Open API 3:


```
{
  "openapi" : "3.0.3",
  "info" : {
    "title" : "Daraa",
    "description" : "This is a test dataset used in the Open Portrayal Framework thread in the OGC Testbed-15 as well as the OGC Vector Tiles Pilot Phase 2.
The data is based on OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.\n\n_Note: This API is based on
API building blocks (e.g., operations, query parameters, or headers) specified in OGC API Standards or drafts of those standards. For more information about
OGC API Standards, see [https://ogcapi.ogc.org](https://ogcapi.ogc.org/). Some building blocks of this API can be preliminary and may change in this API,
because they are not yet based on a stable specification. The maturity is stated for each building block._",
    "contact" : {
      "name" : "interactive instruments GmbH",
      "email" : "mail@interactive-instruments.de"
    },
    "license" : {
      "name" :
"The dataset was provided by the US National Geospatial Intelligence Agency (NGA) for development, testing and demonstrations in initiatives of the Open
Geospatial Consortium (OGC). For any reuse of the data outside this API, please contact NGA."
    },
    "version" : "1.0.0"
  },
  "servers" : [ {
    "url" : "https://demo.ldproxy.net/daraa"
  } ],
  "tags" : [ ],
  "paths" : {
    "/" : {
      "get" : {
        "tags" : [ "Capabilities" ],
        "summary" : "landing page",
        "description" : "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 other resources in the API.\n\n_Maturity: `STABLE`_",
        "externalDocs" : {
          "description" : "The specification that describes this operation: OGC API - Features - Part 1: Core",
          "url" : "https://docs.ogc.org/is/17-069r4/17-069r4.html"
        },
        "operationId" : "getLandingPage",
        "parameters" : [ {
          "$ref" : "#/components/parameters/fCommon"
        } ],
        "responses" : {
          "200" : {
            "description" : "The operation was executed successfully.",
            "content" : {
              "application/json" : {
                "schema" : {
                  "$ref" : "#/components/schemas/LandingPage"
                }
              },
              "text/html" : {
                "schema" : {
                  "$ref" : "#/components/schemas/htmlSchema"
                }
              }
            }
          },
          "400" : {
            "description" : "Bad Request"
          },
          "406" : {
            "description" : "Not Acceptable"
          },
          "500" : {
            "description" : "Server Error"
          }
        },
        "x-maturity" : "STABLE_OGC"
      }
    },
```


You can access an HTML representation of the API definition here.

> **Note**
>
> The use of `/api` on the server is optional and the API definition may be hosted in a different path or on completely separate server.

Feature collections

Data offered through an implementation of **OGC API - Features - Part 1: Core** is organized into one or more feature collections. The `Collections` resource provides information about and access to the list of collections.

For each collection, there is a link to the detailed description of the collection (represented by the path **/collections/{collectionId}** and link relation **self**).

For each collection, there is a link to the features in the collection (represented by the path **/collections/{collectionId}/items** and link relation **items**) and other information about the collection. The following information is provided by the server to describe each collection:

- A local identifier for the collection that is unique for the dataset
- A list of coordinate reference systems (CRS) in which geometries may be returned by the server
- An optional title and description for the collection
- An optional extent that can be used to provide an indication of the spatial and temporal extent of the collection
- An optional indicator about the type of the items in the collection (the default value, if the indicator is not provided, is `feature`).

Below is an extract from the response to the request https://demo.ldproxy.net/daraa/collections?f=json


```
{
  "title": "Daraa",
  "description": "This is a test dataset used in the Open Portrayal Framework thread in the OGC Testbed-15 as well as the OGC Vector Tiles Pilot Phase 2. The
data is based on OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.",
  "collections": [
    {
      "title": "Aeronautic (Curves)",
      "description": "Aeronautical Facilities: Information about an area specifically designed and constructed for landing, accommodating, and launching
military and/or civilian aircraft, rockets, missiles and/or spacecraft.<br/>Aeronautical Aids to Navigation: Information about electronic equipment, housings,
and utilities that provide positional information for direction or otherwise assisting in the navigation of airborne aircraft.",
      "id": "AeronauticCrv",
      "extent": {
        "spatial": {
          "bbox": [
            [36.395158, 32.693301, 36.430814, 32.717333]
          ],
          "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        },
      "storageCrs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
      "links": [
        {
          "rel": "items",
          "type": "application/geo+json",
          "title": "Access the features in the collection 'Aeronautic (Curves)' as GeoJSON",
          "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items?f=json"
        },
        {
          "rel": "self",
          "title": "The 'Aeronautic (Curves)' feature collection",
          "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv"
        }
      ]
     }
    },
    {
      "title": "Other (Points)",
      "id": "o2s_p",
      "extent": {
        "spatial": {
          "bbox": [
            [35.939604, 32.544963, 36.443695, 32.984648]
          ],
          "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        }
      },
      "storageCrs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
      "links": [
        {
          "rel": "items",
          "type": "application/geo+json",
          "title": "Access the features in the collection 'Other (Points)' as GeoJSON",
          "href": "https://demo.ldproxy.net/daraa/collections/o2s_p/items?f=json"
        },
        {
          "rel": "self",
          "title": "The 'Other (Points)' feature collection",
          "href": "https://demo.ldproxy.net/daraa/collections/o2s_p"
        }
      ],
```

```
    }
  ]
```


Feature collection

The **Collection** resource provides detailed information about the collection identified in a request.

Below is an extract from the response to the request https://demo.ldproxy.net/daraa/collections/AeronauticCrv?f=json


```
{
  "title": "Aeronautic (Curves)",
  "description": "Aeronautical Facilities: Information about an area specifically designed and constructed for landing, accommodating, and launching military and/or civilian aircraft, rockets, missiles and/or spacecraft.<br/>Aeronautical Aids to Navigation: Information about electronic equipment, housings, and utilities that provide positional information for direction or otherwise assisting in the navigation of airborne aircraft.",
  "id": "AeronauticCrv",
  "extent": {
    "spatial": {
      "bbox": [
        [36.395158, 32.693301, 36.430814, 32.717333]
      ],
      "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    },
    "temporal": {
      "interval": [
        [
          "2011-03-16T14:51:12Z",
          "2015-09-11T19:15:35Z"
        ]
      ],
      "trs": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
    }
  },
  "itemType": "feature",
  "crs": [
    "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
    "http://www.opengis.net/def/crs/EPSG/0/3395",
    "http://www.opengis.net/def/crs/EPSG/0/3857",
    "http://www.opengis.net/def/crs/EPSG/0/4326"
  ],
  "storageCrs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
  "links": [
    {
      "rel": "items",
      "type": "application/geo+json",
      "title": "Access the features in the collection 'Aeronautic (Curves)' as GeoJSON",
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items?f=json"
    }
    {
      "rel": "self",
      "type": "application/json",
      "title": "This document",
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv?f=json"
    }
  ],
}
```


Features

The Features resource returns a document consisting of features contained by the collection identified in a request. The features included in the response are determined by the server based on the query parameters of the request. To support access to larger collections without overloading the client, the API supports paged access with links to the next page, if more features are selected than the page size.

Below is an extract from the response to the request https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items?f=json


```
{
  "type": "FeatureCollection",
  "numberReturned": 10,
  "numberMatched": 20,
  "timeStamp": "2023-11-29T08:38:10Z",
  "features": [
    {
      "type": "Feature",
      "id": 1,
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [[[36.4270030, 32.7114540],[36.4251990, 32.7137030]]]
      },
      "properties": {
        "F_CODE": "GB075",
        "ZI001_SDV": "2011-03-16T14:51:12Z",
        "UFI": "2d008c34-4458-4226-b335-cf903d261ce9",
        "ZI005_FNA": "No Information",
```

```
        "FCSUBTYPE": 100454
      }
    },
    {
      "type": "Feature",
      "id": 2,
      "geometry": {
        "type": "MultiLineString",
        "coordinates": [
          [[ 36.4009090, 32.7000770 ],
            [ 36.4031330, 32.7013330 ],
            [ 36.4208880, 32.7113020 ],
            [ 36.4231110, 32.7125400 ],
            [ 36.4251990, 32.7137030 ],
            [ 36.4252970, 32.7137690 ]
          ]
        ]
      },
      "properties": {
        "F_CODE": "GB075",
        "ZI001_SDV": "2015-09-11T19:15:35Z",
        "UFI": "1257bf27-3f91-461d-8a3b-a95af2ea1f5a",
        "ZI005_FNA": "No Information",
        "FCSUBTYPE": 100454
      }
    }]
}
```


Note that this document is a valid GeoJSON document.

Additional parameters may be used to select only a subset of the features in the collection.

A **bbox** or **datetime** parameter may be used to select only the subset of the features in the collection that are within the bounding box specified by the **bbox** parameter or the time interval specified by the **datetime** parameter. An example request that uses the **bbox** parameter is https://demo.ldproxy.net/daraa/collections/VegetationSrf/items?f=json&bbox=36.0832432,32.599852,36.1168237,32.6283697

> **Note**
>
> The effect of the bbox parameter can be easily seen when comparing the HTML response from applying the bbox parameter to the response without any bbox parameter.

The **limit** parameter may be used to control the page size by specifying the maximum number of features that should be returned in the response. An example request that uses the **limit** parameter is https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items?f=json&limit=2

Each page may include information about the number of selected and returned features (`numberMatched` and `numberReturned`) as well as links to support paging (link relation `next`).

Feature

The Feature resource is used for retrieving an individual feature, its geometric representation and other properties. In the example below, the feature with an `id` of 1 is retrieved. The response is retrieved through the request https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=json


```
{
  "type": "Feature",
  "id": 1,
  "geometry": {
    "type": "MultiLineString",
    "coordinates": [
      [
        [
          36.4270030,
          32.7114540
        ],
        [
          36.4251990,
          32.7137030
        ]
      ]
    ]
  },
  "properties": {
    "F_CODE": "GB075",
    "ZI001_SDV": "2011-03-16T14:51:12Z",
```

```
    "UFI": "2d008c34-4458-4226-b335-cf903d261ce9",
    "ZI005_FNA": "No Information",
    "FCSUBTYPE": 100454
  },
  "links": [
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=json",
      "rel": "self",
      "type": "application/geo+json",
      "title": "This document"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=jsonfgc",
      "rel": "alternate",
      "type": "application/vnd.ogc.fg+json;compatibility=geojson",
      "title": "This document as JSON-FG (GeoJSON Compatibility Mode)"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=csv",
      "rel": "alternate",
      "type": "text/csv",
      "title": "This document as CSV"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=fgb",
      "rel": "alternate",
      "type": "application/flatgeobuf",
      "title": "This document as FlatGeobuf"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=html",
      "rel": "alternate",
      "type": "text/html",
      "title": "This document as HTML"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv/items/1?f=jsonfg",
      "rel": "alternate",
      "type": "application/vnd.ogc.fg+json",
      "title": "This document as JSON-FG"
    },
    {
      "href": "https://demo.ldproxy.net/daraa/collections/AeronauticCrv?f=json",
      "rel": "collection",
      "type": "application/json",
      "title": "The collection the feature belongs to"
    }
  ]
}
```


Client usage

In this workshop we'll cover different OGC API - Features client tools two JavaScript libraries (Leaflet and OpenLayers), one desktop GIS (QGIS) and a C++ library (GDAL).

LEAFLET

Leaflet can read GeoJSON out-of-the-box, from a file or an API. As OGC API - Features can expose data as GeoJSON by using `f=json` in the request, the response can be read directly in LeafLet using the following code:

```
fetch('https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=100', {
    headers: {
      'Accept': 'application/geo+json'
    }
  }).then(response => response.json())
  .then(data => {
  L.geoJSON(data).addTo(map);
});
```

Leaflet also has an external plugin which allows OGC API - Features to be used natively:

```
// Import following in <head> tag
//   <script src='https://unpkg.com/leaflet-featuregroup-ogcapi@0.1.0/Leaflet.FeatureGroup.OGCAPI.js'></script>


var overlay = L.featureGroup.ogcApi("https://demo.ldproxy.net/zoomstack/", {
    collection: "airports",
    limit: 500,
    padding: 0.2
}).addTo(map);
```

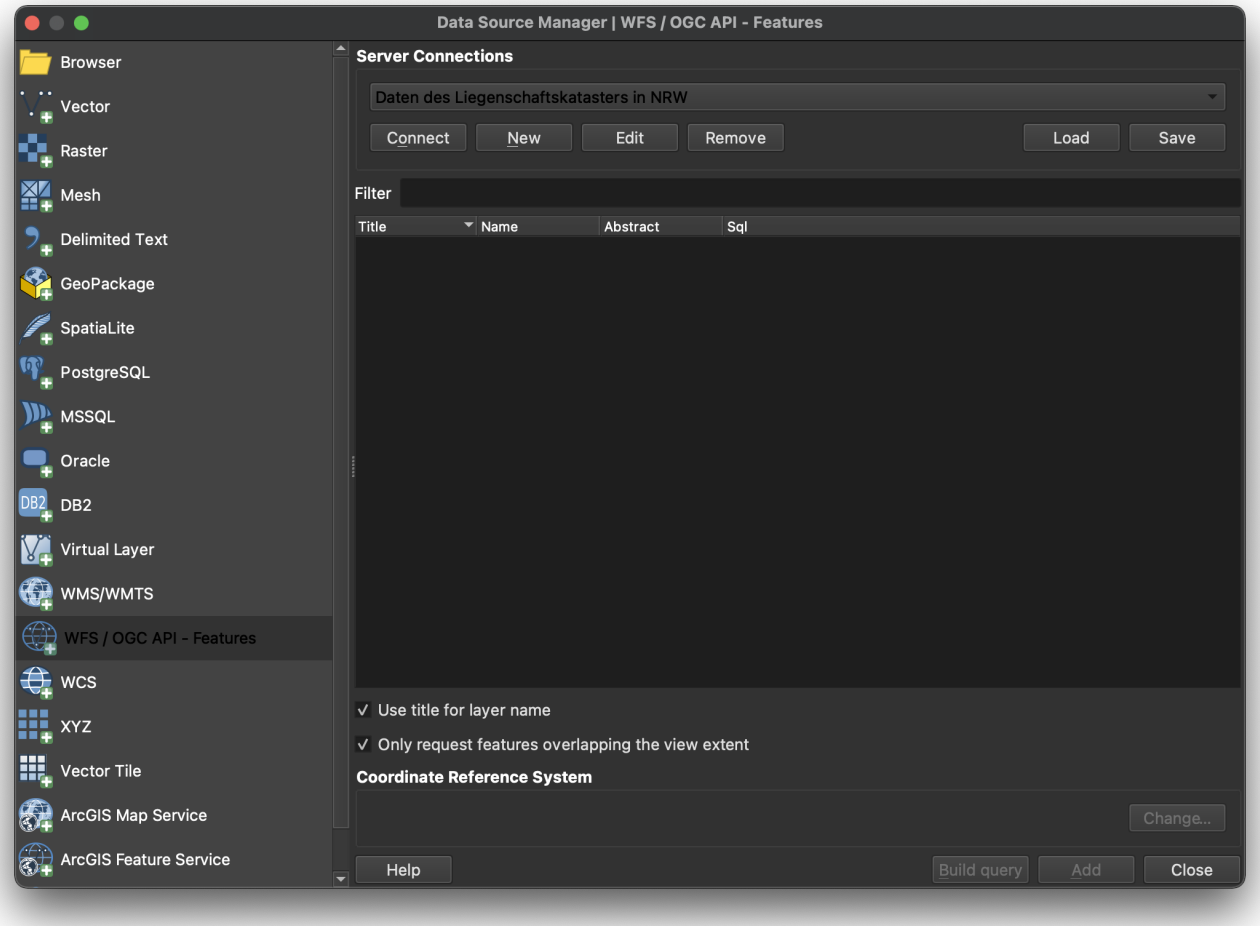
OPENLAYERS

OpenLayers also understands GeoJSON by default. An OGC API - Features response can be consumed using the following code:

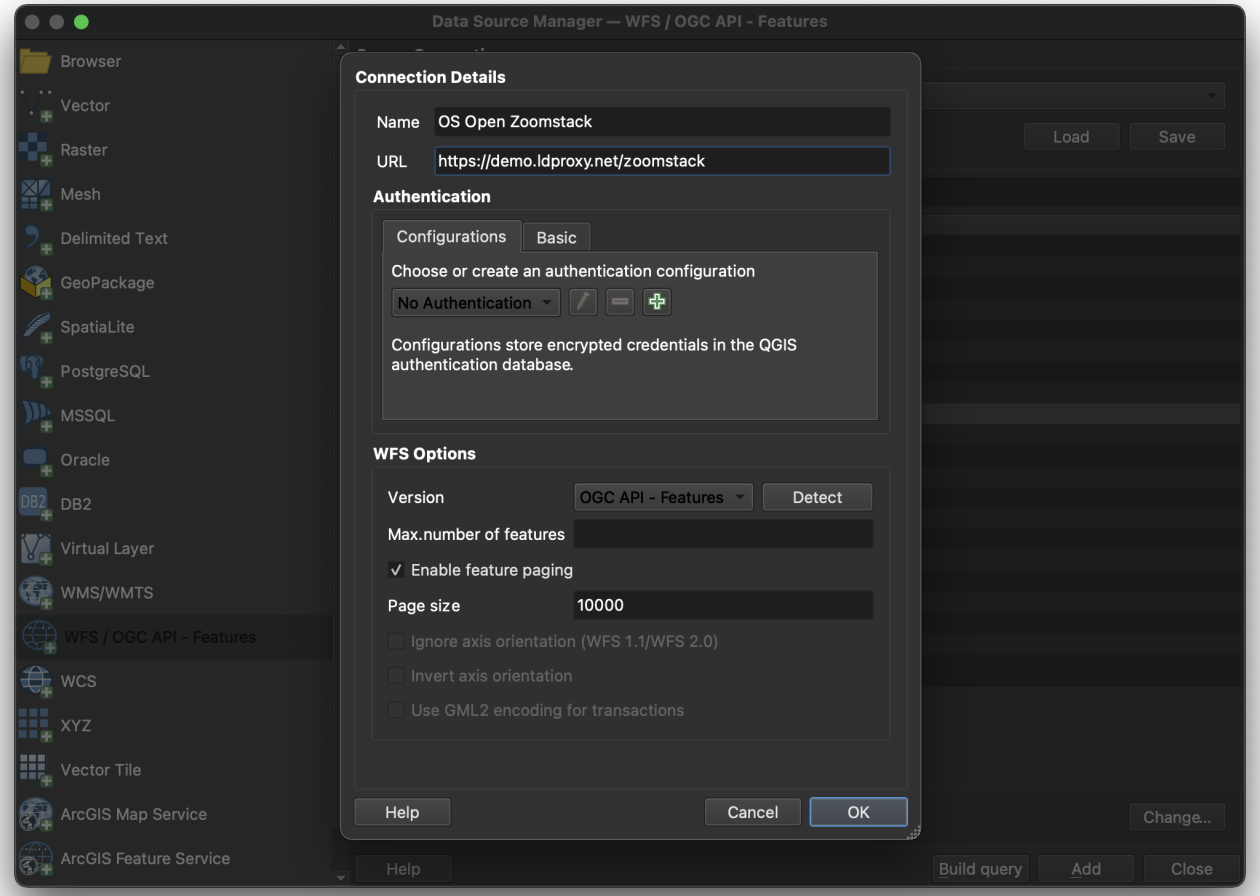
```
fetch('https://demo.ldproxy.net/zoomstack/collections/airports/items?limit=100', {
    headers: {
      'Accept': 'application/geo+json'
    }
  }).then(response => response.json())
  .then(data => {
  map.addLayer(new ol.layer.Vector({
    source: new ol.source.Vector({
      features: new ol.format.GeoJSON().readFeatures(data, { featureProjection: 'EPSG:3857' }),
      attributions: 'Contains OS data &copy; Crown copyright and database right 2021.'
    })
  }));
});
```

QGIS

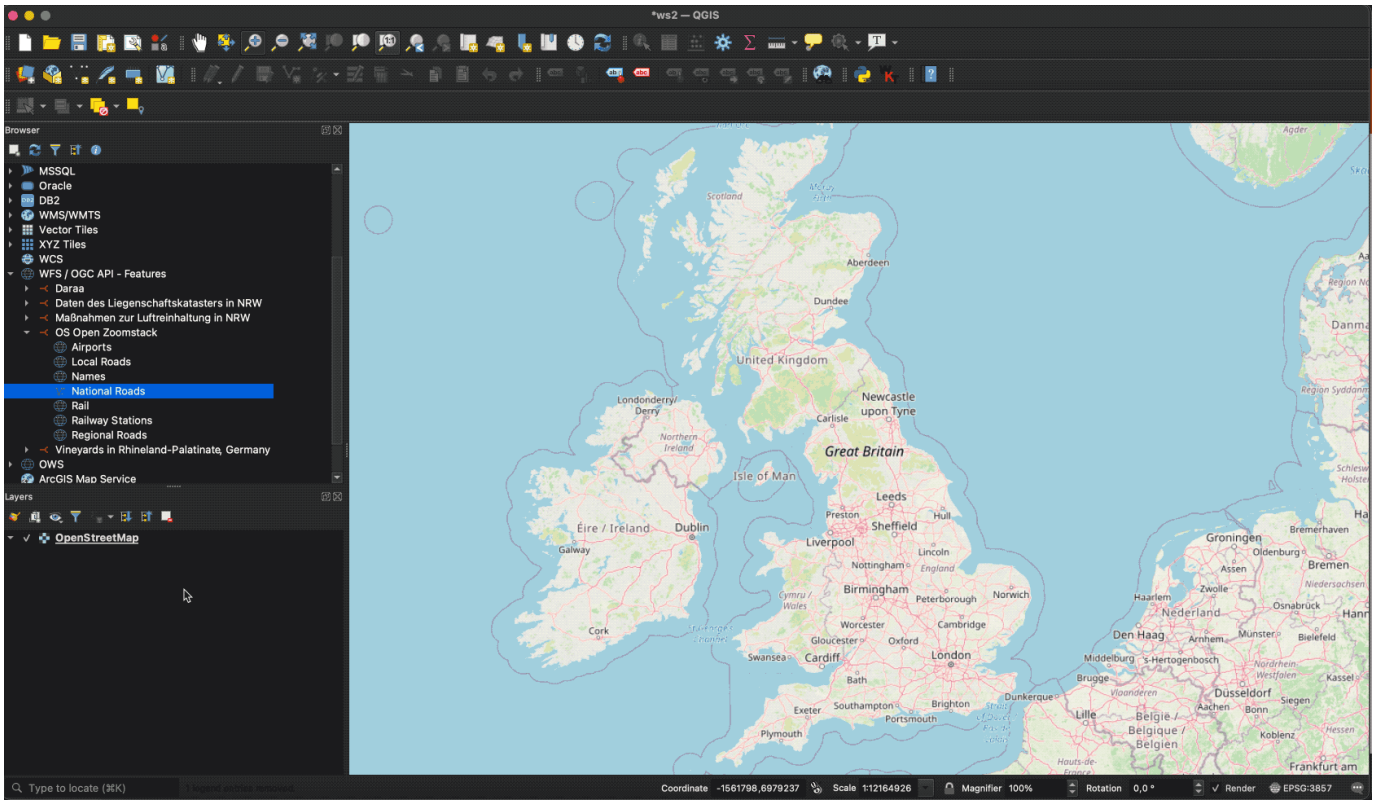
QGIS supports OGC API - Features and WFS using the same vector layer provider. Open the Data Source Manager and go to the "WFS / OGC API Features" tab.



Provide the connection information. The URL is the URL of the OGC API landing page resource (in this case https://demo.ldproxy.net/zoomstack). Make sure "Enable feature paging" is checked.



Note that, if a collection has millions of features and the map view covers the extent of the collection, QGIS will try to load all features. To avoid this, you can, for example, restrict the scale range in which the layer should be visible.



GDAL

GDAL supports OGC API - Features as core vector format. The below example demonstrates usage via `ogrinfo` against an OGC API - Features endpoint:

```
ogrinfo OAPIF:https://demo.ldproxy.net/zoomstack
INFO: Open of `OAPIF:https://demo.ldproxy.net/zoomstack'
      using driver `OAPIF' successful.
1: airports (title: Airports) (Point)
2: boundaries (title: Boundaries) (Line String)
3: contours (title: Contours) (Line String)
4: district_buildings (title: District Buildings) (Polygon)
5: etl (title: ETL) (Line String)
6: foreshore (title: Foreshore) (Polygon)
7: greenspace (title: Greenspace) (Polygon)
8: land (title: Land) (Polygon)
9: local_buildings (title: Local Buildings) (Polygon)
10: names (title: Names) (Point)
11: national_parks (title: National Parks) (Polygon)
12: rail (title: Rail) (Line String)
13: railway_stations (title: RailwayStation) (Point)
14: roads_local (title: Local Roads) (Line String)
15: roads_national (title: National Roads) (Line String)
16: roads_regional (title: Regional Roads) (Line String)
17: sites (title: Sites) (Multi Polygon)
18: surfacewater (title: Surface Water) (Polygon)
19: urban_areas (title: Urban Areas) (Polygon)
20: waterlines (title: Waterlines) (Line String)
21: woodland (title: Woodland) (Polygon)
```

GeoJSON

The OGC API - Features GeoJSON Requirements Class specifies a GeoJSON based encoding for based on RFC7946. Given the ubiquity of GeoJSON, numerous tools exist to validate process and decode/encode GeoJSON, making OGC API - Features GeoJSON easy to include in data processing pipelines. OGC API - Features includes the JSON Schema for the GeoJSON representation and thus can be used for runtime or offline validation of data payloads. Applications based on OGC API - Features GeoJSON can extend and constrain the schema accordingly for domain specific workflows.

10.4.3 Summary

OGC API - Features provides functionality for working with vector data on the Web. This deep dive provided an overview of the standard and the various Resources and endpoints that are supported, as well as example of how-to access it using different clients.

10.5 OGC API - Tiles

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Tiles standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Tiles standard is
- Describe what can be done with OGC API - Tiles implementations
- Understand the main resources offered by OGC API - Tiles implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Tiles implementation
- Understand how to issue requests to an implementation of OGC API - Features
- Be able to find an OGC API - Tiles endpoint and use it through a client

10.5.1 Introduction

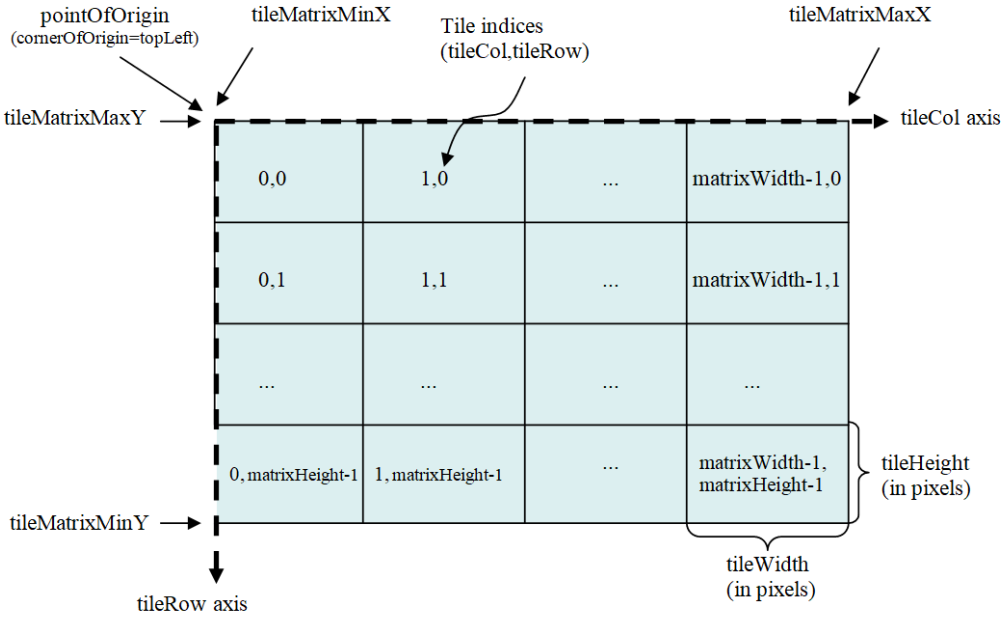
OGC API - Tiles is a standard that defines building blocks for creating Web APIs that support the retrieval of geospatial information as tiles. Different forms of geospatial information are supported, such as tiles of vector features ("vector tiles"), coverages, maps (or imagery) and other types of geospatial information. Although it can be used independently, the OGC API - Tiles building blocks can be combined with other OGC API Standards and draft specifications for additional capabilities or increasing interoperability for specific types of data. The OGC API - Tiles standard references the OGC Two Dimensional Tile Matrix Set (TMS) and Tileset Metadata standard, which defines logical models and encodings for specifying tile matrix sets and describing tile sets.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Tiles - Part 1: Core** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Tiles - Part 1: Core** standard for additional detail.

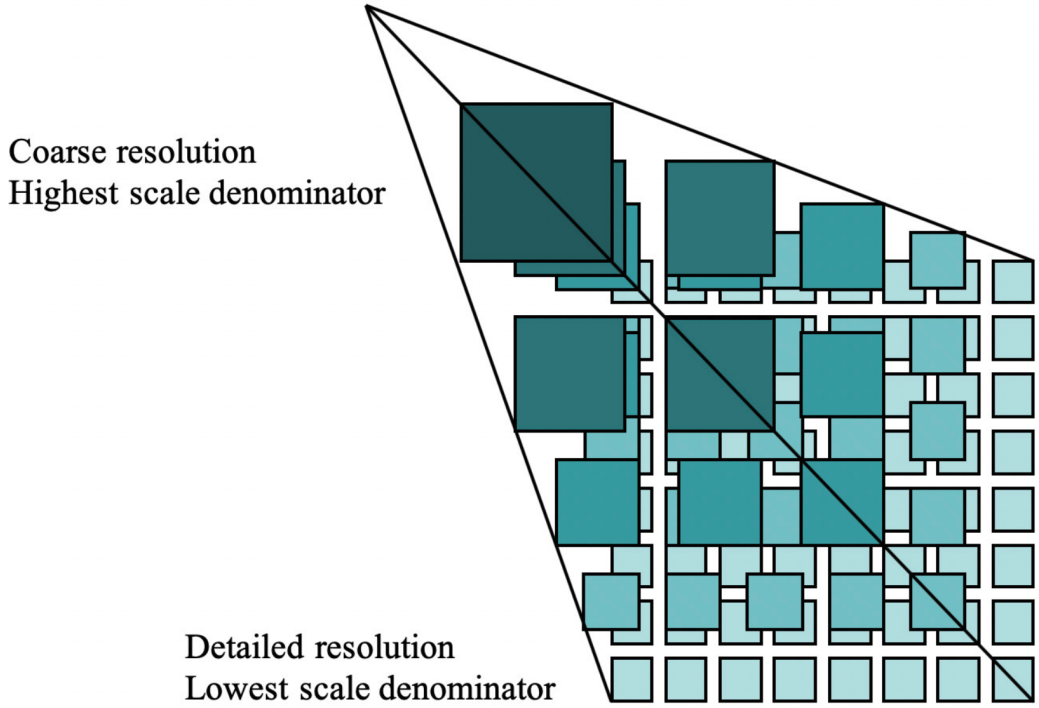
These concepts are at the core of this standard:

- **Tiling Scheme:** schema used to partitioning the space into individual tiles, potentially featuring multiple levels of detail. A tiling scheme is usually defined on top of a CRS, althought it can use other spatial reference systems.
- **Tile Matrix:** tiling grid in a given 2D coordinate reference system, associated to a specific scale and partitioning space (e.g.:



tiling scheme).

- **Tile Matrix Set:** tiling scheme consisting of a set of tile matrices defined at different scales covering approximately the same area and having a common coordinate reference system. A Tile Matrix has a unique alphanumeric identifier in the Tile Matrix Set. Some tile-based implementations prefer to use the zoom level number.



- **Tile Set:** set of tiles resulting from tiling data according to a particular tiling scheme.

Note

- A tile matrix can be implemented as a set of image files (e.g., PNG or JPEG) in a file folder, each file representing a single tile.
- In some standards the Tile Matrix Set concept is called an *image pyramid*.

Background

History

The OGC API - Tiles standard is a successor to the OGC's Web Map Tile Service (WMTS) standard, focusing on simple reusable REST API building blocks which can be described using the OpenAPI specification. Whereas WMTS focused on map tiles, the OGC API - Tiles standard has been designed to support any form of tiled data.

Versions

OGC API - Tiles - Part 1: Core version 1.0.0 is the current latest version

Test suite

A test suite is available for:

- OGC API - Tiles - Part 1

Implementations

Implementations can be found on the implementations page.

USAGE

There are at least two ways to approach an implementation of the OGC API - Tiles Standard.

- Read the landing page, look for links, follow them and discover new links until the desired resource is found
- Read a Web API definition document that specifies a list of paths and path templates to resources.

Once you have discovered the relevant resources, then retrieve the list of available tiling schemes from the resource `/tileMatrixSets` to identify the tiling scheme of interest. Retrieve the details of the specific tiling scheme with `/tileMatrixSets/{tileMatrixSetId}`.

Once you have identified a tiling scheme of interest, you can retrieve tile set metadata for that tiling scheme through `/tiles/{tileMatrixSetId}` and also retrieve individual tiles with `/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

RELATION TO OTHER OGC STANDARDS

Although the OGC API - Tiles Standard is designed as a building block that can be leveraged by (or with) other OGC API Standards adding precisions about specific types of data available as tiles (e.g., OGC API - Features standard, and OGC API - Maps and OGC API - Coverages candidate standards), the conformance classes defined in this Standard are still concrete enough to make it possible to support distributing and requesting various types of tiled data, including coverages, vector features and maps, by relying strictly on the content herein and in the OGC Two Dimensional Tile Matrix Set and Tile Set Metadata 2.0 standard.

Overview of Resources

OGC API - Tiles - Part 1: Core defines the resources listed in the following table.

Resource	Method	Path
Landing page	GET	/
Conformance declaration	GET	/conformance
API definition	GET	/api
Tile matrix sets	GET	/tileMatrixSets
Tile matrix set	GET	/tileMatrixSets/{tileMatrixSetId}
Dataset tileset	GET	/tiles
Dataset tileset metadata	GET	/tiles/{tileMatrixSetId}
Dataset feature tile	GET	/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}
Map tileset list	GET	/map/tiles
Map tileset metadata	GET	/map/tiles/{tileMatrixSetId}
Map tile	GET	/map/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}
Collections	GET	/collections
Collection	GET	/collections/{collectionId}
Feature tileset list	GET	/collections/{collectionId}/tiles
Feature tileset metadata	GET	/collections/{collectionId}/tiles/{tileMatrixSetId}
Feature tile	GET	/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}
Map tileset list	GET	/collections/{collectionId}/map/tiles
Map tileset metadata	GET	/collections/{collectionId}/map/tiles/{tileMatrixSetId}
Map tile	GET	/collections/{collectionId}/map/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}
Coverage tileset list	GET	/collections/{collectionId}/coverage/tiles
Coverage tileset metadata	GET	/collections/{collectionId}/coverage/tiles/{tileMatrixSetId}
Coverage tile	GET	/collections/{collectionId}/coverage/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}

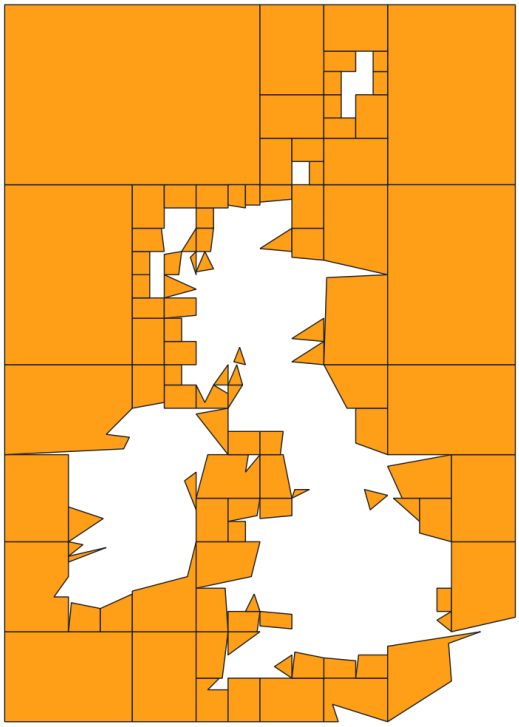
Example

This demonstration server publishes tiled feature data through an interface that conforms to OGC API - Tiles.

An example request that can be used to retrieve data, referenced to WebMercatorQuad, from the OS Zoomstack collection is https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad/0/0/0?f=mvt

In this case the data is encoded in Mapbox Vector Tiles (MVT) format.

Once downloaded, a client application can then display or process the data.



10.5.2 Resources

Landing page

Given OGC API - Tiles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Conformance declarations

Given OGC API - Tiles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

API Definition

Given OGC API - Tiles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Collections

Given OGC API - Tiles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Collection

Given OGC API - Tiles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Tiling Schemes

This endpoint retrieves a list of links to the descriptions of the tile matrix sets supported by the OGC Web API. These could be one or many of the well-known tile matrix sets listed in Annex D of OGC Two Dimensional Tile Matrix Set and Tile Set Metadata, or custom ones.

As an example, we can see an extract of the response to this request: https://demo.ldproxy.net/daraa/tileMatrixSets?f=json


```
  "tileMatrixSets": [
    {
      "title": "Google Maps Compatible for the World",
      "id": "WebMercatorQuad",
      "uri": "http://www.opengis.net/def/tilematrixset/OGC/1.0/WebMercatorQuad",
      "links": [
        {
          "rel": "self",
          "title": "Tile matrix set 'WebMercatorQuad'",
          "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WebMercatorQuad"
        }
      ]
    },
    {
      "title": "CRS84 for the World",
      "id": "WorldCRS84Quad",
      "uri": "http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldCRS84Quad",
      "links": [
        {
          "rel": "self",
          "title": "Tile matrix set 'WorldCRS84Quad'",
          "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WorldCRS84Quad"
        }
      ]
    },
    {
      "title": "World Mercator WGS84 (ellipsoid)",
      "id": "WorldMercatorWGS84Quad",
      "uri": "http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldMercatorWGS84Quad",
      "links": [
        {
          "rel": "self",
          "title": "Tile matrix set 'WorldMercatorWGS84Quad'",
          "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WorldMercatorWGS84Quad"
        }
      ]
    }
  ]
```


If we append the tile matrix set id to this url, we will get the description of one specific tile matrix set, as we can see in the example below, generated with this request:

https://demo.ldproxy.net/daraa/tileMatrixSets/WebMercatorQuad?f=json


```
{
  "title": "Google Maps Compatible for the World",
  "id": "WebMercatorQuad",
  "crs": "http://www.opengis.net/def/crs/EPSG/0/3857",
  "wellKnownScaleSet": "http://www.opengis.net/def/wkss/OGC/1.0/GoogleMapsCompatible",
  "uri": "http://www.opengis.net/def/tilematrixset/OGC/1.0/WebMercatorQuad",
  "tileMatrices": [
    {
      "id": "0",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 1,
      "matrixHeight": 1,
      "scaleDenominator": 559082264.028717,
      "cellSize": 156543.033928041,
      "pointOfOrigin": [
        -20037508.3427892,
        20037508.3427892
      ],
      "cornerOfOrigin": "topLeft"
    },
    {
      "id": "1",
      "tileWidth": 256,
      "tileHeight": 256,
      "matrixWidth": 2,
      "matrixHeight": 2,
      "scaleDenominator": 279541132.014358,
      "cellSize": 78271.5169640204,
      "pointOfOrigin": [
        -20037508.3427892,
        20037508.3427892
      ],
      "cornerOfOrigin": "topLeft"
    },
  }
```


Note that apart from the descriptive metadata, the response also contains a detailed list of available tile matrices.

Dataset Tilesets

These endpoints define how a list of tilesets can be associated to an OGC API dataset / landing page.

For vector tiles, we can request tiles using the `/tiles` endpoint. As an example, this is part of the response triggered with this request:

https://demo.ldproxy.net/daraa/tiles?f=json


```
{
  "title": "Daraa",
  "description": "This is a test dataset used in the Open Portrayal Framework thread in the OGC Testbed-15 as well as the OGC Vector Tiles Pilot Phase 2. The
data is based on OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.",
  "tilesets": [
    {
      "links": [
        {
          "rel": "self",
          "title": "Access the data as tiles in the tile matrix set 'WebMercatorQuad'",
          "href": "https://demo.ldproxy.net/daraa/tiles/WebMercatorQuad"
        },
        {
          "rel": "http://www.opengis.net/def/rel/ogc/1.0/tiling-scheme",
          "title": "Definition of the tiling scheme",
          "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WebMercatorQuad"
        },
        {
          "rel": "item",
          "type": "application/vnd.mapbox-vector-tile",
          "title": "Mapbox vector tiles; the link is a URI template where {tileMatrix}/{tileRow}/{tileCol} is the tile in the tiling scheme
'WebMercatorQuad'",
          "href": "https://demo.ldproxy.net/daraa/tiles/WebMercatorQuad/{tileMatrix}/{tileRow}/{tileCol}?f=mvt",
          "templated": true
        }
      ],
```


We can request metadata about a particular tileset by appending the tile matrix set ID: `/tiles/{tileMatrixSetId}`. For instance, the example below is triggered by this request:

https://demo.ldproxy.net/daraa/tiles/WebMercatorQuad?f=json


```
{
  "tilejson": "3.0.0",
  "tiles": [
    "https://demo.ldproxy.net/daraa/tiles/WebMercatorQuad/{z}/{y}/{x}?f=mvt"
  ],
  "vector_layers": [
    {
      "id": "AeronauticCrv",
      "fields": {
        "id": "Integer",
        "F_CODE": "String",
        "ZI001_SDV": "String",
        "UFI": "String",
        "ZI005_FNA": "String",
        "FCSUBTYPE": "Integer",
        "ZI006_MEM": "String",
        "ZI001_SDP": "String"
      },
      "description": "",
      "maxzoom": 18,
      "minzoom": 6,
      "geometry_type": "lines"
    },
```


Finally we can request the actual data, in this case a vector tile, using `/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`.

We can reuse the same endpoints for map or coverage tiles, but in those cases we need to introduce `map` or `coverage` in the path.

Map tileset list:

- `/map/tiles`

Map tileset metadata:

- `/map/tiles/{tileMatrixSetId}`

Map tile:

- `/map/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

GeoData Tilesets

These endpoints define how a list of tilesets can be associated to an OGC API collection.

For vector tiles, you can retrieve the tileset list of a given collection with `/collections/{collectionId}/tiles`. For instance, the sample below is extracted from the response to this request:

https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles?f=json


```
{
  "title": "Structure (Surfaces)",
  "tilesets": [
    {
      "links": [
        {
          "rel": "self",
          "title": "Access the data as tiles in the tile matrix set 'WebMercatorQuad'",
          "href": "https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad"
        },
        {
          "rel": "http://www.opengis.net/def/rel/ogc/1.0/tiling-scheme",
          "title": "Definition of the tiling scheme",
          "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WebMercatorQuad"
        },
        {
          "rel": "item",
          "type": "application/vnd.mapbox-vector-tile",
          "title": "Mapbox vector tiles; the link is a URI template where {tileMatrix}/{tileRow}/{tileCol} is the tile in the tiling scheme
'WebMercatorQuad'",
          "href": "https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad/{tileMatrix}/{tileRow}/{tileCol}?f=mvt",
          "templated": true
        }
      ],
```


The tileset metadata of a specific tile matrix set, can be retrieved by appending the tile matrix set ID: `/collections/{collectionId}/tiles/{tileMatrixSetId}`. For instance, the following response was extracted from this request:

https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad?f=json


```
  "links": [
    {
      "rel": "self",
      "type": "application/json",
      "title": "This document",
      "href": "https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad?f=json"
    },
    {
      "rel": "alternate",
      "type": "application/vnd.mapbox.tile+json",
      "title": "This document as TileJSON",
      "href": "https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad?f=tilejson"
    },
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/tiling-scheme",
      "title": "Definition of the tiling scheme",
      "href": "https://demo.ldproxy.net/daraa/tileMatrixSets/WebMercatorQuad"
    },
    {
      "rel": "item",
      "type": "application/vnd.mapbox-vector-tile",
      "title": "Mapbox vector tiles; the link is a URI template where {tileMatrix}/{tileRow}/{tileCol} is the tile in the tiling scheme
'{{tileMatrixSetId}}'",
      "href": "https://demo.ldproxy.net/daraa/collections/StructureSrf/tiles/WebMercatorQuad/{tileMatrix}/{tileRow}/{tileCol}?f=mvt",
      "templated": true
    }
  ],
  "dataType": "vector",
  "tileMatrixSetId": "WebMercatorQuad",
  "tileMatrixSetURI": "http://www.opengis.net/def/tilematrixset/OGC/1.0/WebMercatorQuad",
  "tileMatrixSetLimits": [
    {
      "tileMatrix": "6",
      "minTileRow": 25,
      "maxTileRow": 25,
      "minTileCol": 38,
      "maxTileCol": 38,
      "numberOfTiles": 1
    },
    {
      "tileMatrix": "7",
      "minTileRow": 51,
      "maxTileRow": 51,
      "minTileCol": 76,
      "maxTileCol": 76,
      "numberOfTiles": 1
    },
```

Finally we can request the actual data, in this case a vector tile, using `/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`.

Similarly to dataset tilesets, we can reuse the same endpoints for map or coverage tiles, but in those cases we need to introduce `map` or `coverage` in the path.

Map tileset list:

- `/collections/{collectionId}/map/tiles`

Map tileset metadata:

- `/collections/{collectionId}/map/tiles/{tileMatrixSetId}`

Map tile:

- `/collections/{collectionId}/map/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

You can see here an example of a request for a (map) tileset list and here an example of a request for (map) tileset metadata.

Client usage

In this section we will demonstrate how-to access OGC API - Tiles using the OpenLayers client.

OPENLAYERS

The latest versions of OpenLayers supports both OGC Vector tiles and Map Tiles, with the `OGCVectorTile` and the `OGCMapTile` classes.

An example of this can be seen on the example page on the OpenLayers website.

```
import MVT from 'ol/format/MVT.js';
import Map from 'ol/Map.js';
import OGCVectorTile from 'ol/source/OGCVectorTile.js';
import VectorTileLayer from 'ol/layer/VectorTile.js';
import View from 'ol/View.js';

const map = new Map({
  target: 'map',
  layers: [
    new VectorTileLayer({
      source: new OGCVectorTile({
        url: 'https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad',
        format: new MVT(),
      }),
      background: '#d1d1d1',
      style: {
        'stroke-width': 0.6,
        'stroke-color': '#8c8b8b',
        'fill-color': '#f7f7e9',
      },
    }),
  ],
  view: new View({
    center: [0, 0],
    zoom: 1,
  }),
});
```

This example shows both, Map and Vector tiles, that do not use the WGS84 CRS.

10.5.3 Summary

OGC API - Tiles specifies a standard for Web APIs that provide tiles of geospatial information. Different forms of geospatial information are supported, such as tiles of vector features ("vector tiles"), coverages, maps (or imagery) and potentially eventually additional types of tiles of geospatial information. This deep dive provided an overview of the standard and the various Resources and endpoints that are supported. It also shows an example of how-to access an OGC API - Tiles endpoint, using a JavaScript client.

10.6 OGC API - Maps

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Maps standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Maps standard is
- Describe what can be done with OGC API - Maps implementations
- Understand the main resources offered by OGC API - Maps implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Maps implementation
- Understand how to issue requests to an implementation of OGC API - Maps
- Be able to find an OGC API - Maps endpoint and use it through a client

10.6.1 Introduction

OGC API - Maps is a standard that describes an API that presents data as maps by applying a style. The standard allows a client application to request maps as images, or change parameters such as size and coordinate reference systems at the time of request, making them implementer-friendly and easily understandable by developers without geospatial experience.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Maps - Part 1: Core** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Maps - Part 1: Core** standard for additional detail.

Background

History

OGC API - Maps standard work was started in 2019. It has been developed in relation to OGC API - Tiles in support of providing both dynamic maps and map tiles.

Versions

OGC API - Maps - Part 1: Core version 1.0.0 is the current latest version

Test suite

There are no test suites currently implemented; they will be made available once the specification is approved, and an executable test suite (ETS) is made availabe as per of OGC CITE.

Implementations

Implementations can be found on the implementations page.

USAGE

RELATION TO OTHER STANDARDS

OGC Web Map Service Interface Standard (WMS): The WMS standard is a long standing and arguably the most well known and utilized OGC standard.

more appropriate when working with client applications that only support classic OGC Web Services. Note as well that WFS adopts the Geography Markup Language (GML) as a default data format. In contrast, OGC API - Features includes recommendations to support HTML and GeoJSON as encodings, where practical. Implementations of OGC API - Features may also optionally support GML, as well as other vector formats.

Overview of Resources

OGC API - Maps - Part 1: Core defines the resources listed in the following table.

> **Note**
>
> This deep dive focuses on the "Collection Maps" Requirement Class of OGC API - Maps. "Dataset Maps" is not included at this time.

Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
API definition	GET	/api	This resource provides metadata about the API itself. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.
Collections	GET	/collections	This resource lists the collections that are offered through the API.
Collection	GET	/collections/{collectionId}	This resource describes the collection identified in the path.
Collection maps in the default style	GET	/collections/{collectionId}/map	This resource presents the map associated with the collection using the default style.
Collection maps	GET	/collections/{collectionId}/styles/{styleId}/map	This resource presents the map associated with the collection using an applicable style.

Example

This demonstration server publishes geospatial data through an interface that conforms to OGC API - Maps.

An example request that can be used to retrieve data from the MapServer WMS demo collection is https://demo.pygeoapi.io/master/collections/mapserver_world_map/map?f=png

Note that given the scope and purpose of OGC API - Maps, the response to the request is a raw PNG image and not raw data.

10.6.2 Resources

Landing page

Given OGC API - Maps uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Conformance declarations

Given OGC API - Maps uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

API Definition

Given OGC API - Maps uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Collections

Given OGC API - Maps uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

OGC API - Maps collection descriptions provide a number of optional properties, including:

- data type: a description of the underlying data supplied by the map (vector, coverage, map)
- min and max scale denominator: minimum and maximum scale denominator for usage of the collection as a map

Below is an extract from the response to the request https://demo.pygeoapi.io/master/collections?f=json.


```
{
  "id": "mapserver_world_map",
  "title": "MapServer demo WMS world map",
  "description": "MapServer demo WMS world map",
  "keywords": [
    "MapServer",
    "world map"
  ],
  "links": [
    {
      "type": "text/html",
      "rel": "canonical",
      "title": "information",
      "href": "https://demo.mapserver.org",
      "hreflang": "en-US"
    },
    {
      "type": "application/json",
      "rel": "root",
      "title": "The landing page of this server as JSON",
      "href": "https://demo.pygeoapi.io/master?f=json"
    },
    {
      "type": "text/html",
      "rel": "root",
      "title": "The landing page of this server as HTML",
      "href": "https://demo.pygeoapi.io/master?f=html"
    },
    {
      "type": "application/json",
      "rel": "self",
      "title": "This document as JSON",
      "href": "https://demo.pygeoapi.io/master/collections/mapserver_world_map?f=json"
    },
    {
      "type": "application/ld+json",
      "rel": "alternate",
      "title": "This document as RDF (JSON-LD)",
      "href": "https://demo.pygeoapi.io/master/collections/mapserver_world_map?f=jsonld"
    },
    {
      "type": "text/html",
      "rel": "alternate",
      "title": "This document as HTML",
      "href": "https://demo.pygeoapi.io/master/collections/mapserver_world_map?f=html"
    },
    {
      "type": "image/png",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/map",
      "title": "Map as png",
      "href": "https://demo.pygeoapi.io/master/collections/mapserver_world_map/map?f=png"
    }
  ],
  "extent": {
    "spatial": {
      "bbox": [
        [
          -180,
          -90,
```

```
          180,
          90
        ]
      ],
      "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    }
  }
}
```

Note

An HTML representation can be viewed if changing `f=html` or not specifying the `f` parameter when working throgh a web browser.

In the `links` array, notice the link with the link relation (`rel`) of `http://www.opengis.net/def/rel/ogc/1.0/map` . This link relation informs the client that the link is an OGC API - Maps interface that provides either a default map (`href`) or a map with various query parameters applied.

Collection

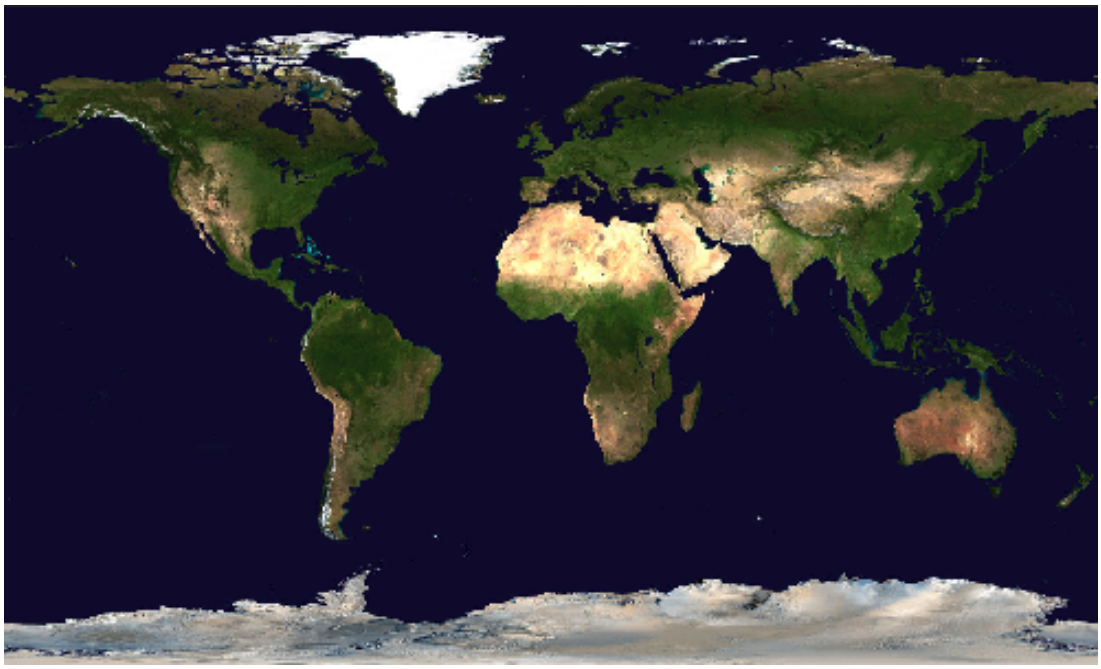
Given OGC API - Maps uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation, as well as the Collections description.

To inspect the specific collection, run the request https://demo.pygeoapi.io/master/collections/mapserver_world_map?f=json.

Collection maps in the default style

Let generate a map from the collection using the link in the above extract:

https://demo.pygeoapi.io/master/collections/mapserver_world_map/map



The request above asks the OGC API - Maps server to generate a default map as determined by the server. In this case, the default is a map of the world with a pixel width of 500 and height of 300.

Additional parameters can be added to the map URL with specific width, height and area of interest.

To clip the map to a desired area of interest (for example, India), use the **bbox** parameter:

https://demo.pygeoapi.io/master/collections/mapserver_world_map/map?f=png&bbox=69,7,99,37

To adjust the map's dimensions, use the **width** and **height** parameters:

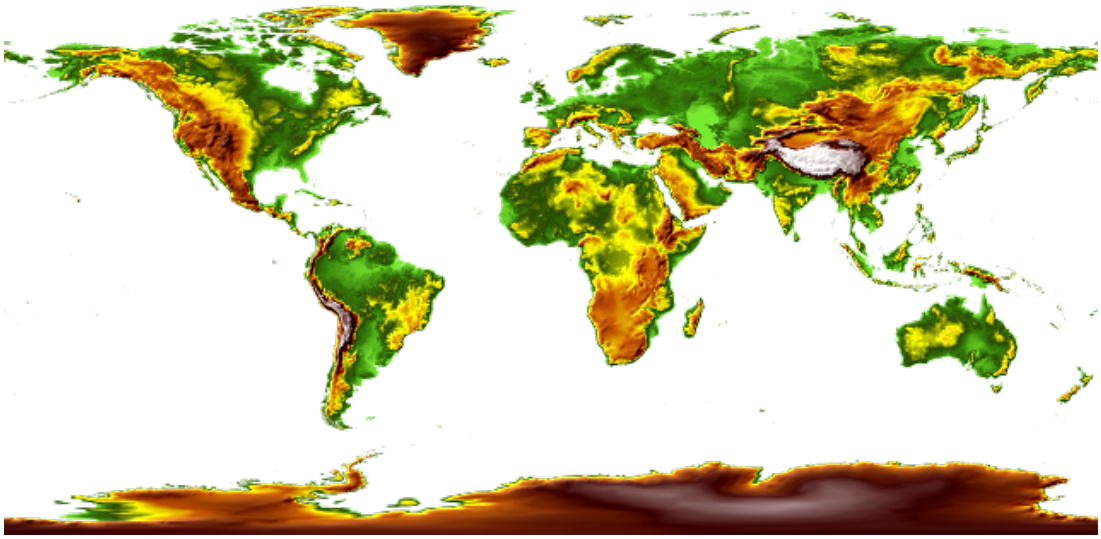
https://demo.pygeoapi.io/master/collections/mapserver_world_map/map?f=png&bbox=69,7,99,37&width=800&height=600

Collection maps

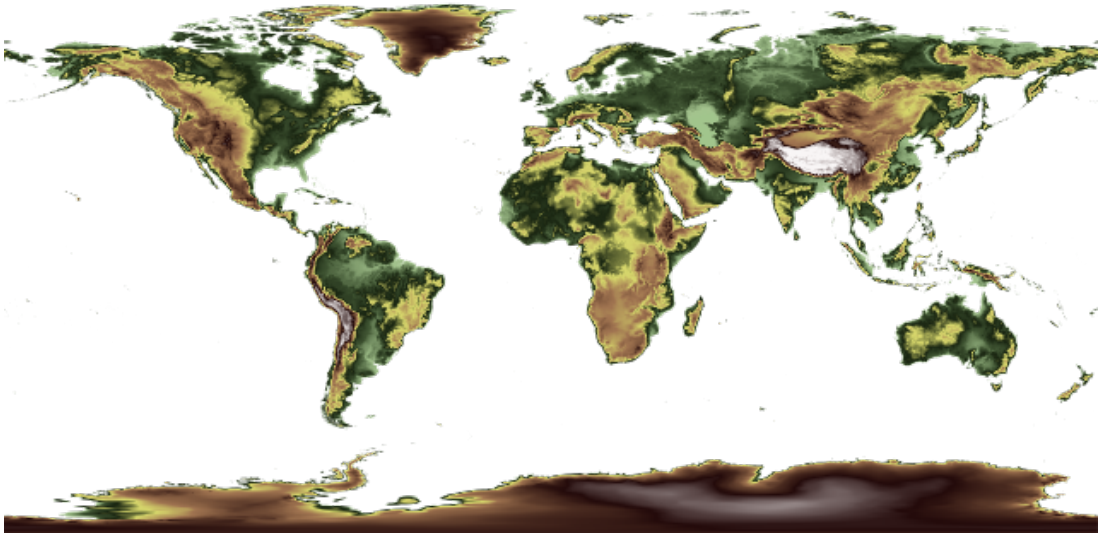
To demonstrate an OGC API - Maps implementation, this demonstration server provides a list of styles for a given dataset at https://test.cubewerx.com/cubewerx/cubeserv/demo/ogcapi/Foundation/collections/gtopo30/styles?f=json.

Each style within the collection can then be requested as a map as follows (using the `colorShaded` and `desaturated` styles):

https://test.cubewerx.com/cubewerx/cubeserv/demo/ogcapi/Foundation/collections/gtopo30/styles/colorShaded/map



https://test.cubewerx.com/cubewerx/cubeserv/demo/ogcapi/Foundation/collections/gtopo30/styles/desaturated/map



10.6.3 Summary

The OGC API - Maps standard describes an API that presents data as maps by applying a style. This deep dive provided an overview of the standard and the various Resources and endpoints that are supported.

10.7 OGC API - Styles

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Styles standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Styles standard is
- Describe what can be done with OGC API - Styles implementations
- Understand the main resources offered by OGC API - Styles implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Styles implementation
- Understand how to issue requests to an implementation of OGC API - Styles
- Be able to find an OGC API - Styles endpoint and use it through a client

10.7.1 Introduction

OGC API - Styles is a standard that describes an API that enables map servers, clients as well as visual style editors, to manage and fetch styles. The styles consist of symbolizing instructions that can be applied by a rendering engine on features and/or coverages. The API implements the conceptual model for style encodings and style metadata.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Styles - Part 1: Core** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Styles - Part 1: Core** standard for additional detail.

Background

History

The need for users and software to be able to control the visual portrayal of geospatial data was already present in the first generation of OGC Web Services.

In 2001, Web Map Service (WMS) 1.1.0 introduced enhanced support for styles using the Styled Layer Descriptor (SLD) Implementation Specification. This specification extended WMS to allow user-defined symbolization of feature data. In 2007, SLD became a profile of WMS.

The evolution of the new web APIs also brought new capabilities in terms of changing, sharing and rendering styles, which were explored in OGC Testbed-15 Open Portrayal Framework (OPF), in 2019. This work was documented in the OGC Testbed-15: Styles API Engineering Report. The following year, in 2020, the charter for the OGC API - Styles Standards Working Group was drafted.

Versions

OGC API - Styles - Part 1: Core is currently in draft.

Test suite

There are no test suites currently implemented; they will be made available once the specification is approved, and an executable test suite (ETS) is made availabe as per of OGC CITE.

Implementations

Implementations can be found on the implementations page.

USAGE

The Styles API supports three main types of consumers:

- Visual style editors that create, update and delete styles for datasets shared by other OGC APIs that publish feature or coverage data. Feature data is either accessed directly or organized into spatial partitions (e.g.: vector tiles).
- OGC API - Maps implementations, that fetch styles and render spatial data (features or coverages) on the server.
- Map clients that fetch styles and render spatial data (features or coverages) on the client.

The draft Standard also defines a conceptual model for styles, style encodings and style metadata. The model defines three main concepts, which are mapped to resources and documents.

- **Style**: the main resource.
- **Stylesheets**: the representation of a style in an encoding like OGC SLD 1.0 or Mapbox Style. Each style is available in one or more stylesheets. Clients will use the stylesheet of a style that fits best based on the capabilities of available tools and their preferences.
- **Style metadata**: general descriptive information about the style, structural information (e.g., layers and attributes), and so forth to allow users to discover and select existing styles for their data. For each style there is style metadata available.

> **Note**
>
> *Draft* **OGC API - Styles - Part 1**: Core offers conformance classes for fetching styles and style metadata, managing and validating styles.

RELATION TO OTHER STANDARDS

OGC API - Styles is designed to be combined with other OGC API Standards, in order to produce styled geospatial data. Feature or coverage data published through OGC APIs, can be styled on client side with styles produced by OGC API - Styles editors. This is the case of OGC API - Features, OGC API - Coverages or OGC API - Tiles (vector tiles). OGC API - Maps has support for fetching styles and rendering geospatial data (features or coverages) on server side.

The styles themselves can be represented using different encodings. As usual in OGC API, no encodings are prescribed, although the draft Standard offers conformance classes for OGC SLD 1.0/1.1 and Mapbox Styles.

Overview of Resources

OGC API - Styles - Part 1: Core defines the resources listed in the following table.

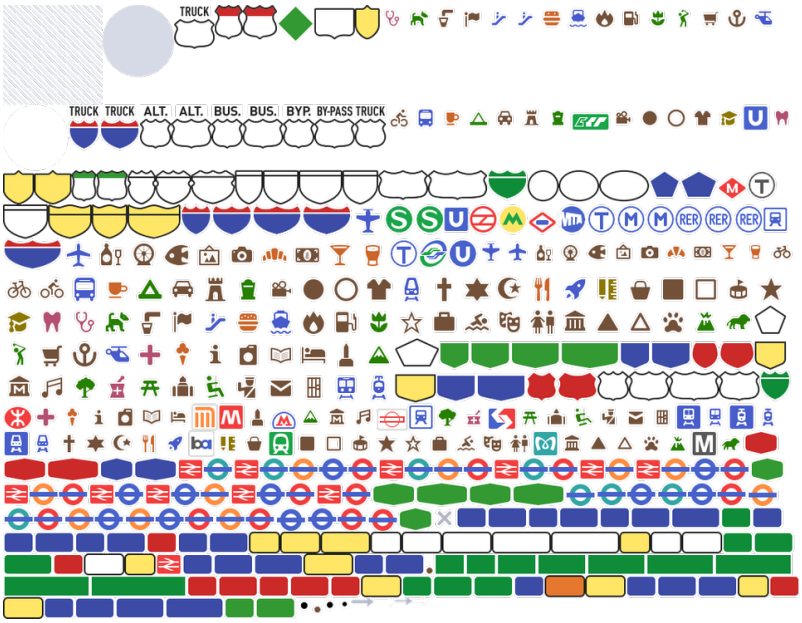
Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
Fetch Styles	GET	/styles	This resource lists the styles that are offered through the API.
Create or validate Styles	POST	/styles	This resource enables the creation of a new Style or the validation of an existing one.
Fetch Style	GET	/styles/{styleId}	This resource retrieves the style identified in the path.
Create or validate Style	PUT	/styles/{styleId}	This resource can be used to update, create or validate the style identified in the path.
Delete Style	DELETE	/styles/{styleId}	This resource can be used to delete the style identified in the path.
Fetch Style metadata	GET	/styles/{styleId}/metadata	This resource retrieves the metadata of the style identified in the path.
Replace Style metadata	PUT	/styles/{styleId}	This resource replaces the metadata of the style identified in the path.
Patch Style metadata	PATCH	/styles/{styleId}	This resource updates parts of the metadata of the style identified in the path.
Fetch resources	GET	/resources	This operation fetches the set of resources (e.g.: symbols and sprites) that have been created and that may be used by reference in stylesheets.
Fetch symbol resource by id	GET	/resources/{resourceId}	This operation fetches the resource with identifier resourceId. The set of available resources can be retrieved at /resources.
Replace symbol resource or add new one	PUT	/resources/{resourceId}	This operation replaces an existing resource with the id resourceId. If no such resource exists, a new resource with that id is added. This operation is only available to registered style authors.
Delete symbol resource	DELETE	/resources/{resourceId}	This operation deletes an existing resource with the id resourceId. If no such resource exists, an error is returned. This operation is only available to registered style authors.

Note

A *sprite* used in a Mapbox Style stylesheet consists of three resources:

- PNG bitmap image (resourceId ends in '.png').
- JSON index file (resourceId of the same name, but ends in '.json' instead of '.png')
- PNG bitmap image for high-resolution displays (the file ends in '.@2x.png').

Each of the resources needs to be created (and eventually deleted) separately.



Example

This demonstration server publishes styles through an interface that conforms to OGC API - Features.

An example request that can be used to list the styles from the Daara collection is https://demo.ldproxy.net/daraa/styles?f=html

Note that the response to the request is HTML in this case.

Alternatively, the same data can be retrieved in GeoJSON format, through the request https://demo.ldproxy.net/daraa/styles?f=json

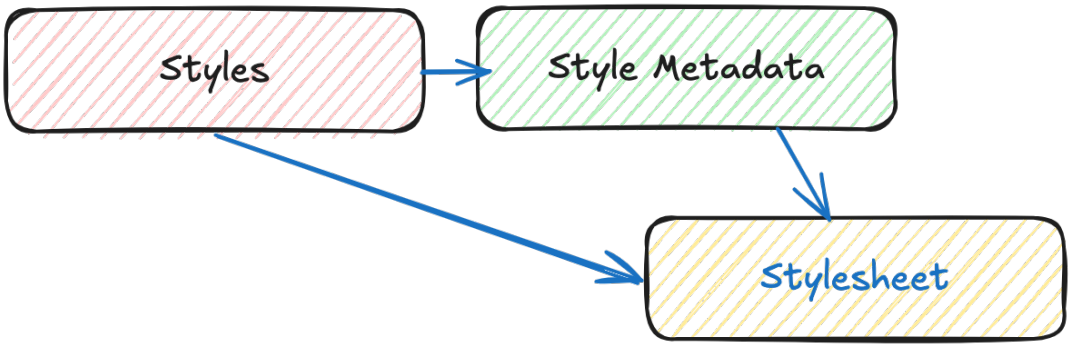
These styles can be rendered by a client application, or applied directly by other OGC APIs that support styles. The example below shows the *Night* style, being applied by OGC API - Maps. https://demo.ldproxy.net/daraa/styles/night?f=html#12.24/32.6264/36.1033

10.7.2 Resources

Styles are the main resources of this API.

- For each style there is style metadata available, with general descriptive information about the style, structural information (e.g., layers and attributes), and so forth to allow users to discover and select existing styles for their data.
- Each style is available as one or more stylesheets - the representation of a style in an encoding like OGC SLD 1.0 or Mapbox Style. Clients will use the stylesheet of a style that fits best based on the capabilities of available tools and their preferences.

A basic request workflow could look like the diagram below, where a client requests the list of styles, and then asks for more information about a particular style, before fetching the stylesheet. In alternative, a client can request the stylesheet directly after the styles request.



Note

This section will focus on the resources related to requirements class "Core": fetching styles, style and style metadata.

Landing page

Given OGC API - Styles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Conformance declarations

Given OGC API - Styles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

API Definition

Given OGC API - Styles uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Style list

This endpoint, lists the styles available on the server, and for each describes basic information like its id, title and description, as well as the available stylesheets.

Below is an extract from the response to the request https://demo.ldproxy.net/daraa/styles?f=json.


```
{
  "styles": [
    {
      "title": "night",
      "id": "night",
      "links": [
        {
          "rel": "describedby",
          "title": "Style metadata",
          "href": "https://demo.ldproxy.net/daraa/styles/night/metadata"
        },
        {
          "rel": "stylesheet",
          "type": "text/html",
          "title": "Web map using the style",
          "href": "https://demo.ldproxy.net/daraa/styles/night?f=html"
        },
        {
          "rel": "stylesheet",
          "type": "application/vnd.mapbox.style+json",
          "title": "Style in format 'Mapbox'",
          "href": "https://demo.ldproxy.net/daraa/styles/night?f=mbs"
        }
      ]
    },
```

In this response, we can see that the links to retrieve more information about the style (e.g.: style metadata) and to retrieve it as a stylesheet.

Style metadata

Requests the metadata for a particular style, so that a client has more information about a potential style of interest. The response format (typically HTML or JSON, but extensions can easily supply others) is determined using HTTP content negotiation.

In the sample below, we request information about the *topographic* style. The full response can be retrieved using this request: https://demo.ldproxy.net/daraa/styles/topographic/metadata?f=json


```
{
  "title": "topographic",
  "links": [
    {
      "rel": "self",
      "type": "application/json",
      "title": "This document",
      "href": "https://demo.ldproxy.net/daraa/styles/topographic/metadata?f=json"
    },
    {
      "rel": "alternate",
      "type": "text/html",
      "title": "This document as HTML",
      "href": "https://demo.ldproxy.net/daraa/styles/topographic/metadata?f=html"
    }
  ],
  "id": "topographic",
  "scope": "style",
  "stylesheets": [
    {
      "title": "Mapbox",
      "version": "8",
      "specification": "https://docs.mapbox.com/mapbox-gl-js/style-spec/",
      "native": true,
      "link": {
        "rel": "stylesheet",
        "type": "application/vnd.mapbox.style+json",
        "title": "Style in format 'Mapbox'",
        "href": "https://demo.ldproxy.net/daraa/styles/topographic?f=mbs"
      }
    }
  ],
```


Fetch Style

This request returns a stylesheet. If multiple encodings are available, the style encoding is determined using HTTP content negotiation. For instance, a client looking for a Mapbox stylesheet, could request the `application/vnd.mapbox.style+json` type.

In the example below, the *topographic* style is retrieved as a Mapbox stylesheet.

https://demo.ldproxy.net/daraa/styles/topographic?f=mbs

This sample shows an extract of the Mapbox spec 8.0, response.


```
    "daraa": {
      "type": "vector",
      "tiles": [
        "https://demo.ldproxy.net/daraa/tiles/WebMercatorQuad/{z}/{y}/{x}?f=mvt"
      ],
      "bounds": [
        35.755073,
        32.357351,
        37.205276,
        33.26714
      ],
      "scheme": "xyz",
      "maxzoom": 16
    }
  },
  "sprite": "https://demo.ldproxy.net/daraa/resources/sprites",
  "glyphs": "https://go-spatial.github.io/carto-assets/fonts/{fontstack}/{range}.pbf",
  "layers": [
    {
      "id": "Grey Background",
      "type": "background",
      "layout": {
        "visibility": "visible"
      },
```

```
      "paint": {
        "background-color": "#d3d3d3"
      }
    },
    {
      "id": "OSM",
      "type": "raster",
      "source": "osm",
      "layout": {
        "visibility": "none"
      }
    },
    {
      "id": "agriculturesrf",
      "type": "fill",
      "source": "daraa",
      "source-layer": "AgricultureSrf",
      "paint": {
        "fill-color": "#7ac5a5"
      }
    },
    {
      "id": "vegetationsrf",
      "type": "fill",
      "source": "daraa",
      "source-layer": "VegetationSrf",
      "paint": {
        "fill-color": "#C2E4B9"
      }
    },
    {
      "id": "settlementsrf.1",
      "type": "line",
      "source": "daraa",
      "source-layer": "SettlementSrf",
      "paint": {
        "line-color": "#000000",
        "line-width": 2
      }
    },
    {
      "id": "settlementsrf.2",
      "type": "fill",
      "source": "daraa",
      "source-layer": "SettlementSrf",
      "paint": {
        "fill-color": "#E8C3B2"
      }
    },
    {
      "id": "militarysrf",
      "type": "fill",
      "source": "daraa",
      "source-layer": "MilitarySrf",
      "paint": {
        "fill-color": "#f3602f",
        "fill-opacity": 0.5
      }
    },
    {
      "id": "culturesrf",
      "type": "fill",
      "source": "daraa",
      "source-layer": "CultureSrf",
      "paint": {
        "fill-color": "#ab92d2",
        "fill-opacity": 0.5
      }
    },
    {
      "id": "hydrographycrv",
      "type": "line",
      "source": "daraa",
      "source-layer": "HydrographyCrv",
      "filter": [
        "==",
        "BH140",
        [
          "get",
          "F_CODE"
        ]
      ],
      "paint": {
        "line-color": "#00A0C6",
        "line-width": [
          "step",
          [
            "zoom"
          ],
          1,
          8,
          2,
          13,
          4
```

```
        ]
      }
    },
```

10.7.3 Summary

The OGC API - Styles candidate Standard describes an API for accessing and managing styles for rendering geospatial data on the web. It provides building blocks for interacting with styles in multiple style encodings and with metadata for the styles. This deep dive provided an overview of the candidate Standard and the various resources and endpoints that are supported.

10.8 OGC API - Processes

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Processes standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Processes standard is
- Describe what can be done with OGC API - Processes implementations
- Understand the main resources offered by OGC API - Processes implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Processes implementation
- Understand how to issue requests to an implementation of OGC API - Processes
- Be able to find an OGC API - Processes endpoint and use it through a client

10.8.1 Introduction

OGC API -- Processes is a standard that supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application. The standard specifies a processing interface to communicate over a RESTful protocol using JavaScript Object Notation (JSON) encodings. The standard leverages concepts from the OGC Web Processing Service (WPS) 2.0 Interface Standard but does not require implementation of a WPS. The Core part of the standard is called **OGC API - Processes - Part 1: Core**. The Core part of the standard supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application either synchronously or asynchronously. Examples of computational processes that can be supported by implementations of this specification include raster algebra, geometry buffering, constructive area geometry, routing, imagery analysis and several others.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Processes - Part 1: Core** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Processes - Part 1: Core** standard for additional detail.

Background

History

Several of the concepts specified in OGC API - Processes originated in work specifying a RESTful interface for WPS 2.0. From February 2019 onwards, all work relating to a RESTful interface for the WPS2.0 was changed to focus on OGC API - Processes.

Versions

OGC API - Processes - Part 1: Core version 1.0.0 is the current latest version

Test suite

Test suites are available for:

- OGC API - Processes - Part 1

All of the test suites are available from the OGC Validator.

Implementations

Implementations can be found on the implementations page.

USAGE

OGC API - Processes - Part 1: Core supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application. Government agencies, private organisations and academic institutes use the OGC API - Processes standard to provide implementations of geospatial algorithms that process data. The benefit of this is that the processing of geospatial data, including data from sensors, can be distributed thereby allowing for more capacity to process larger amounts of data.

In addition to the approved part above, The OGC API - Processes Standards Working Group (SWG) is working on the following drafts:

- *Draft* **OGC API - Processes - Part 2: Deploy, Replace, Undeploy** extends the core capabilities specified in Part 1 with the ability to dynamically add, modify and/or delete individual processes using an implementation (endpoint) of the OGC API - Processes Standard.
- *Draft* **OGC API - Processes - Part 3: Workflows and Chaining** extends the core capabilities specified in Part 1 with the ability to chain nested processes, refer to both local and external processes and collections of data accessible via OGC API standards as inputs to a process, and trigger execution of processes through OGC API data delivery specifications such as OGC API — Tiles, DGGS, Coverages, Features, EDR and Maps.

RELATION TO OTHER OGC STANDARDS

- OGC Web Processing Service Interface Standard (WPS): The WPS Standard provides a standard interface that simplifies the task of making simple or complex computational geospatial processing services accessible via web services. The OGC API --- Processes Standard is a newer and more modern way of programming and interacting with resources over the web while allowing better integration into existing software packages. The OGC API --- Processes Standard addresses all of the use cases that were addressed by the WPS Standard, while also leveraging the OpenAPI specification and a resource-oriented approach.

Overview of Resources

OGC API - Processes - Part 1: Core defines the resources listed in the following table.

Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
API definition	GET	/api	This resource provides metadata about the API itself. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.
Process list	GET	/processes	Process identifiers, links to process descriptions.
Process description	GET	/processes/{processID}	Retrieves a process description.
Process execution	POST	/processes/{processID}/execution	Creates and executes a job.
Job status info	GET	/jobs/{jobID}	Retrieves information about the status of a job.
Job results	GET	/jobs/{jobID}/results	Retrieves the resul(s) of a job.
Job list	GET	/jobs	Retrieves the list of jobs.
Job deletion	DELETE	/jobs/{jobID}	Cancels and deletes a job.

Example

This demonstration server offers and executes various processes through an interface that conforms to OGC API - Processes.

An example request that can be used to browse all the available processes can be found at https://demo.pygeoapi.io/master/processes.

Note that the response to the request is HTML in this case.

Alternatively, the same data can be retrieved in GeoJSON format, through the request https://demo.pygeoapi.io/master/processes?f=json

10.8.2 Resources

Landing page

Given OGC API - Processes uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

Conformance declarations

Given OGC API - Processes uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

API Definition

Given OGC API - Processes OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Process list

Processes offered through an implementation of **OGC API - Processes** are organized into one or more processes. The `/processes` endpoint provides information about and access to the list of processes.

For each process, there is a link to the detailed description of the process (represented by the path **/processes/{processId}** and link relation **self**). In addition, there are links for executing the process as well as the list of jobs as a results of executing the process.

Process information also includes whether the process can be run in synchronous and / or asynchronous mode (job control options). Asynchronous mode is valuable for executing long running jobs without blocking the HTTP request/response workflow. This also means the client can check back for the status of the job as well as the result once it is completed.

Finally, there are definitions for the input structure required to run the process (expressed as JSON Schema), as well as the output structure a client should expect when receiving a response from the process execution.

Below is an extract from the response to the request https://demo.pygeoapi.io/master/processes?f=json


```
{
    "version": "0.2.0",
    "id": "hello-world",
    "title": "Hello World",
    "description": "An example process that takes a name as input, and echoes it back as output. Intended to demonstrate a simple process with a single
literal input.",
    "jobControlOptions":[
        "sync-execute",
        "async-execute"
    ],
    "keywords":[
        "hello world",
        "example",
        "echo"
    ],
    "links":[
        {
            "type": "text/html",
            "rel": "about",
            "title": "information",
            "href": "https://example.org/process",
            "hreflang": "en-US"
        },
        {
            "type": "application/json",
            "rel": "self",
            "href": "https://demo.pygeoapi.io/master/processes/hello-world?f=json",
            "title": "Process description as JSON",
            "hreflang": "en-US"
        },
        {
            "type": "text/html",
            "rel": "alternate",
            "href": "https://demo.pygeoapi.io/master/processes/hello-world?f=html",
            "title": "Process description as HTML",
            "hreflang": "en-US"
        },
        {
            "type": "text/html",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list",
            "href": "https://demo.pygeoapi.io/master/jobs?f=html",
            "title": "jobs for this process as HTML",
            "hreflang": "en-US"
        },
        {
            "type": "application/json",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list",
            "href": "https://demo.pygeoapi.io/master/jobs?f=json",
            "title": "jobs for this process as JSON",
            "hreflang": "en-US"
        },
        {
            "type": "application/json",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/execute",
            "href": "https://demo.pygeoapi.io/master/processes/hello-world/execution?f=json",
            "title": "Execution for this process as JSON",
            "hreflang": "en-US"
        }
    ],
    "inputs":{
        "name":{
            "title": "Name",
            "description": "The name of the person or entity that you wish tobe echoed back as an output",
            "schema":{
                "type": "string"
            },
            "minOccurs":1,
```

```
            "maxOccurs":1,
            "metadata":null,
            "keywords":[
                "full name",
                "personal"
            ]
        },
        "message":{
            "title": "Message",
            "description": "An optional message to echo as well",
            "schema":{
                "type": "string"
            },
            "minOccurs":0,
            "maxOccurs":1,
            "metadata":null,
            "keywords":[
                "message"
            ]
        }
    },
    "outputs":{
        "echo":{
            "title": "Hello, world",
            "description": "A \"hello world\" echo with the name and (optional) message submitted for processing",
            "schema":{
                "type": "object",
                "contentMediaType": "application/json"
            }
        }
    },
    "example":{
        "inputs":{
            "name": "World",
            "message": "An optional message."
        }
    },
    "outputTransmission":[
        "value"
    ]
}
```


Process description

The previous example demonstrated process information for all processes offered by an OGC API - Processes server. To access process information for a single process, run the below request against the demo server:

https://demo.pygeoapi.io/master/processes/hello-world?f=json

> **Note**
>
> Single process information requires the process identifier as part of the URL

Process execution

Now that we have the appropriate process information, we can execute the process. Process execution requires that requests are run using HTTP POST, with a payload as specified/required by the server (JSON).

> **Note**
>
> Web browsers cannot easily make HTTP POST requests, so we use the curl command. You are welcome to use any tool that is able to execute HTTP POST requests per below.

```
curl -X 'POST' \
  'https://demo.pygeoapi.io/master/processes/hello-world/execution' \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "inputs": {
    "message": "Great to see you here",
    "name": "OGC API workshop participant"
  }
}'
```

The server will respond with an immediate response (synchronous mode by default) as per below:


```
{
    "id": "echo",
    "value": "Hello OGC API workshop participant! Great to see you here"
}
```


To execute the same process in asynchronous mode, we need to add the **Prefer: respond-async** HTTP header. As well, the response to an ascynchronous process execution is always empty, where the HTTP **Location** header contains a URL to the resulting job information.

> **Note**
>
> We add the `-v` option to the curl command below to be able to inspect the response headers

```
curl -v -X 'POST' \
  'https://demo.pygeoapi.io/master/processes/hello-world/execution' \
  -H 'Prefer: respond-async' \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "inputs": {
    "message": "Great to see you here",
    "name": "OGC API workshop participant"
  }
}'
```

An extract of the response shows the **Location** (location) HTTP header:

```
< HTTP/2 201
< access-control-allow-origin: *
< content-language: en-US
< content-type: application/json
< date: Mon, 04 Dec 2023 16:33:06 GMT
< location: https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003
< preference-applied: respond-async
< server: gunicorn
< x-powered-by: pygeoapi 0.16.dev0
< content-length: 4
```

> **Note**
>
> The URL of the `location` HTTP header will always be unique

Job status info

Using the URL from the `location` HTTP header above, we can inspect the status of the job:

https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003?f=json


```
{
    "processID": "hello-world",
    "jobID": "cdbc641c-92c2-11ee-9c88-0242ac120003",
    "status": "successful",
    "message": "Job complete",
    "progress":100,
    "parameters":null,
    "job_start_datetime": "2023-12-04T16:33:06.806485Z",
    "job_end_datetime": "2023-12-04T16:33:06.812615Z",
    "links":[
        {
            "href": "https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003/results?f=html",
            "rel": "about",
            "type": "text/html",
            "title": "results of job cdbc641c-92c2-11ee-9c88-0242ac120003 as HTML"
        },
        {
            "href": "https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003/results?f=json",
            "rel": "about",
            "type": "application/json",
            "title": "results of job cdbc641c-92c2-11ee-9c88-0242ac120003 as JSON"
        }
```

```
    ]
}
```

Job results

Here we see that the job is fully executed and complete, but does not contain the actual results. To inspect the actual results, we use the link objects which provide the results accordingly:

https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003/results?f=json

> **Note**
>
> We see that the the results of the synchronous and asynchronous request/responses are identical, and that only the execution control is different.

Job list

In the same manner that an OGC API - Processes server provides access to process information for all its processes, the server provides the same for all of its jobs (from any process) using the following URL:

https://demo.pygeoapi.io/master/jobs?f=json

Job deletion

If we wish to delete a given job, we can execute an HTTP DELETE request agains the the job ID.

> **Note**
>
> Web browsers cannot easily make HTTP DELETE requests, so we use the curl command. You are welcome to use any tool that is able to execute HTTP DELETE requests per below.

```
curl -X 'DELETE' https://demo.pygeoapi.io/master/jobs/cdbc641c-92c2-11ee-9c88-0242ac120003
```

> **Note**
>
> Try running an HTTP GET on the job that was just deleted and verify that it no longer exists (HTTP 404).

> **Note**
>
> Some servers may implement access control to prevent erroneous or unwanted deletion of a job or other resource.

10.8.3 Summary

The OGC API - Processes standard enables the execution of computing processes and the retrieval of metadata describing the purpose and functionality of the processes. This deep dive provided an introduction to the standard and an overview of its various endpoints, that enable monitoring, creating, updating and deleting those processes on a server.

10.9 OGC API - Records

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Records standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Records standard is
- Describe what can be done with OGC API - Records implementations
- Understand the main resources offered by OGC API - Records implementations
- Understand how to retrieve a description of the capabilities of an OGC API - Records implementation
- Understand how to issue requests to an implementation of OGC API - Records
- Be able to find an OGC API - Records endpoint and use it through a client

10.9.1 Introduction

OGC API - Records is a multi-part draft specification that offers the capability to create, modify, and query metadata on the Web. The draft specification enables the discovery of geospatial resources by standardizing the way collections of descriptive information about the resources (metadata) are exposed. The draft specification also enables the discovery and sharing of related resources that may be referenced from geospatial resources or their metadata by standardizing the way all kinds of records are exposed and managed. Part 1 covers read-only access to records and simple query capabilities. Additional capabilities that address specific needs will be specified in additional parts. Capabilities for richer queries or to create, update or delete records will be specified in additional parts.

Note

OGC API - Records leverages OGC API - Features as a baseline with similar URL endpoints and request/response workflow, for the Searchable Catalog and Local.

Background

History

OGC API - Records standard work was started in 2018 and originally referred to as **OGC CAT4.0**. It has since followed the development of OGC API - Features as a baseline.

Versions

OGC API - Records - Part 1: Core has been submitted to the OGC Architecture Board (OAB) and has completed public review stage. It is expected to be finalized in Q4 2024.

Test suite

There are no test suites currently implemented; they will be made available once the specification is approved, and an executable test suite (ETS) is made availabe as per of OGC CITE.

Implementations

Implementations can be found on the implementations page.

USAGE

OGC API - Records supports 3 main deployment patterns:

- Crawlable catalog: browse and navigation of a set of metadata records via links
- Searchable catalog: API capability to query and filter a collection of metadata records based on serch criteria (bbox, datetime, q, etc.)
- Local resources catalog: searchable catalog functionality applied at the collection level of an API

OGC API - Records also supports a core queryable model. That is, a set of common queryable properties that can be used against any OGC API - Records server regardless of the metadata format/standard and/or the design of the underlying metadata repository.

Note

For the purposes of this deep dive, we will focus on the Searchable catalog deployment pattern.

RELATION TO OTHER STANDARDS

OGC Catalogue Service for the Web (CSW): The CSW standard is more appropriate when working with client applications that only support classic OGC Web Services. Note as well that CSW adopts a core metadata model based on Dublin Core by default. In contrast, OGC API - Records includes recommendations to support HTML and GeoJSON as encodings, where practical. Implementations of OGC API - Records may also optionally support XML metadata formats, such as ISO 19115/19139.

Overview of Resources

OGC API - Records - Part 1: Core defines the resources listed in the following table.

Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
API definition	GET	/api	This resource provides metadata about the API itself. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.
Record collections	GET	/collections	This resource lists the record collections that are offered through the API.
Record collection	GET	/collections/{collectionId}	This resource describes the record collection identified in the path.
Records access	GET	/collections/{collectionId}/items	This resource presents the records that are contained in the collection.
Record core	GET	/collections/{collectionId}/items/{recordId}	This resource presents the record that is identified in the path

As mentioned earlier, OGC API - Records heavily leverages OGC API - Features as a baseline building block. While OGC API - Records allows for any metadata model, a key difference and value add is the ability to describe a core record model and queryables. This allows for interoperability and integration across catalogs to be able to describe geospatial resources in a consistent manner.

For example, a metadata repository can be modelled after the ISO 19115 standard, and be exposed via OGC API - Records by means of "mapping" the ISO elements to the core record model and queryables.

The core record is the atomic unit of information in a catalog. A full description of the core properties of a record can be found in https://docs.ogc.org/DRAFTS/20-004.html#core-properties. The core record is a GeoJSON compatible representation with fixed elements in the `properties` object/block.

Example

This demonstration server publishes metadata geospatial data through an interface that conforms to OGC API - Records.

An example request that can be used to retrieve data from the Sample metadata records from Dutch Nationaal georegister record collection is https://demo.pygeoapi.io/master/collections/dutch-metadata?f=html

Note that the response to the request is HTML in this case.

Alternatively, the same data can be retrieved in GeoJSON format, through the request https://demo.pygeoapi.io/master/collections/dutch-metadata?f=json

A client application can then retrieve the GeoJSON document and display or process it.

10.9.2 Resources

Landing page

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

Conformance declarations

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

API Definition

Given OGC API - Records uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Records collections

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed initial explanation.

OGC API - Records collection descriptions provide the following additional properties:

- A required title for the collection
- A required type for the collection
- A required indicator about the type of the items in the collection (`record`)

Below is an extract from the response to the request https://demo.pygeoapi.io/master/collections?f=json illustrating a single record collection:


```
{
    "id": "dutch-metadata",
    "type": "Catalog",
    "itemType": "record",
    "title": "Sample metadata records from Dutch Nationaal georegister",
    "description": "Sample metadata records from Dutch Nationaal georegister",
    "keywords":[
        "netherlands",
        "open data",
        "georegister"
    ],
    "links":[
```

```
        {
            "type": "application/json",
            "rel": "self",
            "title": "This document as JSON",
            "href": "https://demo.pygeoapi.io/master/collections/dutch-metadata?f=json"
        },
        {
            "type": "application/geo+json",
            "rel": "items",
            "title": "items as GeoJSON",
            "href": "https://demo.pygeoapi.io/master/collections/dutch-metadata/items?f=json"
        }
    ]
}
```


Records collection

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed initial explanation, as well as the Records collections description..

Records access

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

Below is an extract from the response to the request https://demo.pygeoapi.io/master/collections/dutch-metadata/items?f=json


```
{
  "type": "FeatureCollection",
  "numberMatched": 308,
  "numberReturned": 10,
  "features": [
    {
      "id": "35149dfb-31d3-431c-a8bc-12a4034dac48",
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              4.690751953125,
              52.358740234375
            ],
            [
              4.690751953125,
              52.6333984375
            ],
            [
              5.020341796875,
              52.6333984375
            ],
            [
              5.020341796875,
              52.358740234375
            ],
            [
              4.690751953125,
              52.358740234375
            ]
          ]
        ]
      },
      "properties": {
        "created": "2021-12-08",
        "updated": "2022-06-10T01:27:47Z",
        "type": "dataset",
        "title": "Kaartboeck 1635",
        "description": "Data uit kaartboeken van de periode 1635 tot 1775. De kaartboeken werden door het waterschap gebruikt om er op toe te zien dat de eigenaren geen water in beslag namen door demping.\nDe percelen op de kaart zijn naar de huidige maatstaven vrij nauwkeurig gemeten en voorzien van een administratie met de eigenaren. bijzondere locaties van molens werven en beroepen worden in de boeken vermeld. Alle 97 kaarten aan een geven een zeer gedetailleerd beeld van de Voorzaan, Nieuwe Haven en de Achterzaan. De bladen Oost en West van de zaan zijn vrij nauwkeurig. De bladen aan de Voorzaan zijn een schetsmatige weergave van de situatie. De kaart van de Nieuwe Haven si weer nauwkeurig te noemen.",
        "providers": [
          "Team Geo, geo-informatie@zaanstad.nl, Gemeente Zaanstad"
        ],
        "externalIds": [
          {
            "scheme": "default",
            "value": "35149dfb-31d3-431c-a8bc-12a4034dac48"
          }
        ],
        "themes": [
          {
            "concepts": [
              "ARGEOLOGIE",
              "MONUMENTEN",
```

```
              "KADASTER",
              "KAARTBOEK",
              "KAARTBOECK",
              "HISTORIE"
            ]
          }
        ],
        "extent": {
          "spatial": {
            "bbox": [
              [
                4.690751953125,
                52.358740234375,
                5.020341796875,
                52.6333984375
              ]
            ],
            "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
          },
          "temporal": {
            "interval": [
              null,
              null
            ],
            "trs": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
          }
        }
      },
      "links": [
        {
          "href": "https://maps-intern.zaanstad.gem.local/geoserver/wms?SERVICE=WMS",
          "rel": "item",
          "title": "geo:kaartboeck",
          "type": "OGC:WMS"
        },
        {
          "href": "https://maps-intern.zaanstad.gem.local/geoserver/wfs?SERVICE=WFS",
          "rel": "item",
          "title": "geo:kaartboeck",
          "type": "OGC:WFS"
        },
        {
          "href": "https://maps-intern.zaanstad.gem.local/geoserver/wfs?
SERVICE=WFS&version=1.0.0&request=GetFeature&typeName=geo:kaartboeck&outputFormat=csv",
          "rel": "item",
          "type": "download"
        },
        {
          "href": "https://maps-intern.zaanstad.gem.local/geoserver/wfs?
SERVICE=WFS&version=1.0.0&request=GetFeature&typeName=geo:kaartboeck&outputFormat=shape-zip",
          "rel": "item",
          "type": "download"
        }
      ]
    }
```


Note that this document is a valid GeoJSON document.

OGC API - Records supports the same query parameters as specified in OGC API - Features. In addition, OGC API - Records adds a set of core, fixed queryables. An example query based on a "search engine" style search using the **q** parameter is https://demo.pygeoapi.io/master/collections/dutch-metadata/items?f=json&q=biomassa

> **Note**
>
> Consult the OGC API - Records - Part 1: Core specification for more information on core queryables.

Record core

Given OGC API - Records uses OGC API - Common and OGC API - Features as building blocks, please see the OGC API - Features deep dive for a detailed explanation.

GeoJSON

The OGC API - Records GeoJSON Requirements Class specifies a GeoJSON based encoding for Record core, based on RFC7946. Given the ubiquity of GeoJSON, numerous tools exist to validate process and decode/encode GeoJSON, making OGC API - Records GeoJSON easy to include in metadata processing pipelines. OGC API - Records includes the JSON Schema for the GeoJSON representation and thus can be used for runtime or offline validation of metadata payloads. Applications based on OGC API - Records GeoJSON can extend and constrain the schema accordingly for domain specific workflows.

10.9.3 Summary

OGC API - Records provides functionality for working with metadata data on the Web. This deep dive provided an overview of the standard and the various Resources and endpoints that are supported.

10.10 OGC API - Environmental Data Retrieval

Audience

Students that are familiar with web services and APIs, and want to have an overview of OGC API - Environmental Data Retrieval standard

Learning Objectives

At the completion of the module students will be able to:

- Explain what the OGC API - Environmental Data Retrieval standard is
- Describe what can be done with OGC API - Environmental Data Retrieval implementations
- Understand the main resources offered by implementations of OGC API - Environmental Data Retrieval
- Understand how to retrieve a description of the capabilities of an implementation of OGC API - Environmental Data Retrieval
- Understand how to issue requests to an implementation of OGC API - Environmental Data Retrieval
- Be able to find an OGC API - Environmental Data Retrieval endpoint and use it through a client

10.10.1 Introduction

OGC API - Environmental Data Retrieval is a standard that provides a family of lightweight interfaces to access Environmental Data resources. The standard, which is also called the Environmental Data Retrieval (EDR) API, addresses two fundamental operations; discovery and query. Discovery operations allow the API to be interrogated to determine its capabilities and retrieve information (metadata) about this distribution of a resource. This includes the API definition of the server as well as metadata about the Environmental Data resources provided by the server. Query operations allow Environmental Data resources to be retrieved from the underlying data store based upon simple selection criteria, defined by this standard and selected by the client.

Note

This tutorial module is not intended to be a replacement to the actual **OGC API - Environmental Data Retrieval** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC API - Environmental Data Retrieval** standard for additional detail.

Background

History

Version 1.1.0 was published on 2023-07-27.

Versions

OGC API - Environmental Data Retrieval version 1.1.0 is the current latest version

Test suite

A test suite is available for:

- OGC API - Environmental Data Retrieval - Part 1

All of the test suites are available from the OGC Validator.

Implementations

Implementations can be found on the implementations page.

USAGE

OGC API - Environmental Data Retrieval provides a family of lightweight query interfaces to access spatio-temporal data resources by requesting data at a position, within an area, along a trajectory or through a corridor. A spatio-temporal data resource is a collection of spatio-temporal data that can be sampled using the EDR query pattern geometries.

The standard provides a standard interface for requesting vector geospatial data consisting of geographic features and their properties. The benefit of this is that client applications can request source data from multiple implementations of the API, and then render the data for display or process the data further as part of a workflow. The standard enables the data to be accessed consistently with other data. Feature properties encoded using common data types such as text strings, date and time can also be accessed consistently.

RELATION TO OTHER OGC STANDARDS

- OGC API - Features: The EDR API is completely compatible with OGC API - Features - Part 1: Core (OGC 17-069r3), in that it supports Collections and Items. It extends the Collection functionality by allowing 'Instances', a form of 'collection of collections'. The EDR API also supports the retrieval of spatiotemporal data by named location as well as coordinates.
- Moving Features: The Moving Features Standards are concerned with things that move along a trajectory, and simultaneously change their orientation through rigid body rotation. The EDR API does not have the concept of orientation, or foliation or prisms. Moving Features and EDR API do share a common conceptual definition, from ISO, of a Trajectory, but the Moving Features Standards encode trajectories in GML, CSV and Moving Features JSON, whereas the EDR API encodes trajectories in WKT.
- Web Coverage Service (WCS) and Coverage Implementation Schema (CIS): The primary messaging mechanism of the EDR API is JSON, including CoverageJSON, over HTTP(S). Implementations of the EDR API are described using the OpenAPI V3.0 specification. The EDR API is consistent with the WCS and CIS standards but does not require the end user or developer to use the terms Domain and RangeSet. The EDR API can also be used to generate a single query against a collection of coverages, providing the data coordinate reference systems are consistent.
- The OGC SensorThings API: SensorThings API follows OData's specification for requesting entities. In contrast, the EDR API makes use of the OpenAPI V3.0 specification for describing resource paths, query options, JSON schema, and other aspects. Further, the EDR API allows for retrieval of coverage data and HTML responses -- both of which are not supported by the SensorThings API.
- Sensor Observation Service (SOS): The EDR API allows for retrieval of coverage data and HTML responses -- both of which are not supported by the SOS standard. Further, SOS implementations use the GetCapabilities operation for providing descriptions of available resources. In contrast, the EDR API uses OpenAPI definition documents for describing available interfaces.

Overview of Resources

OGC API - Environmental Data Retrieval Standard defines the resources listed in the following table.

Overview of OGC API - EDR resources

Resource	Method	Path	Purpose
Landing page	GET	/	This is the top-level resource, which serves as an entry point.
Conformance declaration	GET	/conformance	This resource presents information about the functionality that is implemented by the server.
API definition	GET	/api	This resource provides metadata about the API itself. Note use of /api on the server is optional and the API definition may be hosted on completely separate server.
Collections metadata	GET	/collections	Metadata describing the collections of data available from this API.
Single Collection metadata	GET	/collections/{collectionId}	Metadata describing the collection of data which has the unique identifier {collectionId}.
Items metadata	GET	/collections/{collectionId}/items	Retrieve metadata about available items.
Query data	GET	/collections/{collectionId}/{queryType}	Retrieve data according to the query pattern
Query instances	GET	/collections/{collectionId}/instances	Retrieve metadata about instances of a collection

Example

This demonstration server publishes environmental data through an interface that conforms to the OGC API - Environmental Data Retrieval standard. A client application is available here .

An example request that can be used to retrieve data from the METAR Observation collection is here .

Note that the response to the request is GeoJSON in this case.

Alternatively, the same data can be retrieved in CoverageJSON format, through this request .

Note that this demonstration server offers data from recent observations, therefore you may need to update the values of the `datetime` parameter to the current day in order to access available METAR observation.

10.10.2 Resources

This section provides basic information about the types of resources that OGC API - Environmental Data Retrieval offers.

Each resource provides **links** that relate to resources. This enables a client application to navigate the resources, from the landing page through to the individual features. The server identifies the relationship between a resource and other linked resources through a **link relation type**, represented by the attribute `rel`. The link relation types used by implementations of the **OGC API - Environmental Data Retrieval** can be found in Section 6.2 of the standard.

Landing page

The landing page is the top-level resource that serves as an entry point. A client application needs to know the location of the landing page of the server. From the landing page, the client application can retrieve links to the Conformance declaration, Collection and API definition paths. An example landing page is at http://labs.metoffice.gov.uk/edr

The link to the API definition is identified through the `service-desc` and `service-doc` link relation types.

The link to the Conformance declaration is identified through the `conformance` link relation type.

The link to the Collections is identified through the `data` link relation type.

An extract from the landing page of a demo server is shown below.


```
{
"title": "Environmental Data Retrevial API concept demonstrator",
"description": "Example EDR API (not for operational use)",
"keywords": [
  "position",
  "area",
  "cube",
  "trajectory",
  "weather",
  "data",
  "api"
],
"terms_of_service": "None",
"provider": {
  "name": "Organization Name",
  "url": "http://example.org"
},
"contact": {
  "email": "you@example.org",
  "phone": "+001-234-567-89",
  "fax": "+001-234-567-89",
  "hours": "Hours of Service",
  "instructions": "During hours of service.  Off on weekends.",
  "address": "Mailing Address",
  "postalcode": "Zip or Postal Code",
  "city": "City",
  "stateorprovince": "Administrative Area",
  "country": "Country"
},
"links": [
  {
    "href": "http://labs.metoffice.gov.uk/edr/api",
    "hreflang": "en",
    "rel": "service-doc",
    "type": "application/vnd.oai.openapi+json;version=3.0",
    "title": "",
    "variables": null
  },
  {
    "href": "http://labs.metoffice.gov.uk/edr/conformance",
    "hreflang": "en",
    "rel": "conformance",
    "type": "application/json",
    "title": "",
    "variables": null
  },
  {
    "href": "http://labs.metoffice.gov.uk/edr/collections",
    "hreflang": "en",
    "rel": "collection",
    "type": "application/json",
    "title": "",
    "variables": null
  }
]
}
```


Conformance declaration

An implementation of OGC API - Environmental Data Retrieval describes the capabilities that it supports by declaring which conformance classes it implements. The Conformance declaration states the conformance classes from standards or community specifications, identified by a URI, that the API conforms to. Clients can then use this information, although they are not required to. Accessing the Conformance declaration using HTTP GET returns the list of URIs of conformance classes implemented by the server. Conformance classes describe the behavior a server should implement in order to meet one or more sets of requirements specified in a standard.

Below is an extract from the response to the request http://labs.metoffice.gov.uk/edr/conformance


```
{
 "conformsTo":[
    "http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core",
    "http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/core",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/oas30",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/html",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/geojson",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/coveragejson",
    "http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/wkt"
 ]
 }
```

API Definition

Given OGC API - Environmental Data Retrieval uses OGC API - Common as a building block, please see the OGC API - Features deep dive for a detailed explanation of an example implementation.

Collections metadata

Data offered through an implementation of **OGC API - Environmental Data Retrevial** is organized into one or more feature collections. The `Collections` resource provides information about and access to the list of collections.

For each collection, there is a link to the detailed description of the collection (represented by the path **/collections/{collectionId}** and link relation **self**).

The following information is provided by the server to describe each collection:

- A local identifier for the collection that is unique for the dataset
- A list of coordinate reference systems (CRS) in which geometries may be returned by the server
- An optional title and description for the collection
- An optional extent that can be used to provide an indication of the spatial and temporal extent of the collection
- An optional indicator about the type of the items in the collection (the default value, if the indicator is not provided, is `feature`).

For each collection, there are links to retrieve data according to supported query patterns (represented by the path **/collections/{collectionId}/{queryType}** and link relation **data**).

For each collection, there is a link to the metadata about items available in the collection (represented by the path **/collections/{collectionId}/items** and link relation **items**) and other information about the collection.

Below is an extract from the response to the request http://labs.metoffice.gov.uk/edr/collections


```
{
  "links": [
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections",
      "hreflang": "en",
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections?f=html",
      "hreflang": "en",
      "rel": "alternate",
      "type": "text/html"
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections?f=xml",
      "hreflang": "en",
      "rel": "alternate",
      "type": "application/xml"
    }
  ],
  "collections": [
    {
      "id": "metar_demo",
      "title": "Metar observations EDR demonstrator",
      "description": "API to access 24 hours of Global Metar Observation data (not for operational use)",
      "keywords": [
        "Metar observation",
        "ICAO identifier",
        "Wind Direction",
        "Wind Speed",
        "Wind Gust",
        "Visibility",
        "Air Temperature",
        "Dew point",
        "Runway Visibility",
        "Weather",
        "Sky condition",
        "Mean Sea Level Pressure",
        "Station Level Pressure",
        "description",
        "restrictions",
        "collection",
        "position",
        "radius",
        "area",
```

```
        "location"
      ],
      "links": [
        {
          "href": "https://www.aviationweather.gov/metar/help",
          "hreflang": "en",
          "rel": "service-doc",
          "type": "text/html",
          "title": ""
        },
        {
          "href": "https://www.weather.gov/disclaimer",
          "hreflang": "en",
          "rel": "restrictions",
          "type": "text/html",
          "title": ""
        },
        {
          "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/",
          "hreflang": "en",
          "rel": "collection",
          "type": "collection",
          "title": ""
        },
        {
          "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/position",
          "hreflang": "en",
          "rel": "data"
        },
        {
          "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/radius",
          "hreflang": "en",
          "rel": "data"
        },
        {
          "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/area",
          "hreflang": "en",
          "rel": "data"
        },
        {
          "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/locations",
          "hreflang": "en",
          "rel": "data"
        }
      ],
      "extent": {
        "spatial": {
          "bbox": [
            -180.0,
            -89.9,
            180.0,
            89.9
          ],
          "crs": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
        },
        "temporal": {
          "interval": [
            "R36/2021-10-03T01:00Z/PT1H"
          ],
          "trs": "TIMECRS[\"DateTime\",TDATUM[\"Gregorian Calendar\"],CS[TemporalDateTime,1],AXIS[\"Time (T)\",future]"
        }
      },
      "data_queries": {
        "position": {
          "link": {
            "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/position",
            "hreflang": "en",
            "rel": "data",
            "variables": {
              "title": "Position query",
              "query_type": "position",
              "output_formats": [
                "CoverageJSON",
                "GeoJSON",
                "IWXXM"
              ],
              "default_output_format": "GeoJSON",
              "crs_details": [
                {
                  "crs": "CRS84",
                  "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
                }
              ]
            }
          }
        },
        "radius": {
          "link": {
            "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/radius",
            "hreflang": "en",
```

```
            "rel": "data",
            "variables": {
              "title": "Radius query",
              "description": "Radius query",
              "query_type": "radius",
              "output_formats": [
                "CoverageJSON",
                "GeoJSON",
                "IWXXM"
              ],
              "default_output_format": "GeoJSON",
              "within_units": [
                "km",
                "miles"
              ],
              "crs_details": [
                {
                  "crs": "CRS84",
                  "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
                }
              ]
            }
          }
        },
        "area": {
          "link": {
            "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/area",
            "hreflang": "en",
            "rel": "data",
            "variables": {
              "title": "Area query",
              "query_type": "area",
              "output_formats": [
                "CoverageJSON",
                "GeoJSON",
                "IWXXM"
              ],
              "default_output_format": "CoverageJSON",
              "crs_details": [
                {
                  "crs": "CRS84",
                  "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
                }
              ]
            }
          }
        },
        "locations": {
          "link": {
            "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/locations",
            "hreflang": "en",
            "rel": "data",
            "variables": {
              "title": "Location query",
              "description": "Location query",
              "query_type": "locations",
              "output_formats": [
                "CoverageJSON",
                "GeoJSON",
                "CSV"
              ],
              "default_output_format": "GeoJSON",
              "crs_details": [
                {
                  "crs": "CRS84",
                  "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
                }
              ]
            }
          }
        }
      },
      "crs": [
        "CRS84"
      ],
      "output_formats": [
        "CoverageJSON",
        "GeoJSON",
        "IWXXM"
      ],
      "parameter_names": {
        "Metar observation": {
          "type": "Parameter",
          "description": "Source Metar observation",
          "unit": {
            "label": "",
            "symbol": {
              "value": "",
```

```
          "type": "http://codes.wmo.int/wmdr/DataFormat/FM-15-metar"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/wmdr/DataFormat/FM-15-metar",
        "label": "Metar observation"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "ICAO identifier": {
      "type": "Parameter",
      "description": "ICAO identifier",
      "unit": {
        "label": "",
        "symbol": {
          "value": "",
          "type": "https://en.wikipedia.org/wiki/ICAO_airport_code"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/bufr4/b/01/_063",
        "label": "ICAO identifier"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Wind Direction": {
      "type": "Parameter",
      "description": "Wind Direction",
      "unit": {
        "label": "degree true",
        "symbol": {
          "value": "\u00b0",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degree"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_aerodromeMeanWindDirection",
        "label": "Wind Direction"
      },
      "measurementType": {
        "method": "mean",
        "period": "-PT10M/PT0M"
      }
    },
    "Wind Speed": {
      "type": "Parameter",
      "description": "Wind Speed",
      "unit": {
        "label": "mph",
        "symbol": {
          "value": "mph",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/mph"
        }
      },
      "observedProperty": {
        "id": " http://codes.wmo.int/common/quantity-kind/aerodromeMeanWindSpeed",
        "label": "Wind Speed"
      },
      "measurementType": {
        "method": "mean",
        "period": "-PT10M/PT0M"
      }
    },
    "Wind Gust": {
      "type": "Parameter",
      "description": "Wind Gust",
      "unit": {
        "label": "mph",
        "symbol": {
          "value": "mph",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/mph"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_aerodromeMaximumWindGustSpeed",
        "label": "Wind Gust"
      },
      "measurementType": {
        "method": "maximum",
        "period": "-PT10M/PT0M"
      }
    },
    "Visibility": {
      "type": "Parameter",
      "description": "Visibility",
      "unit": {
        "label": "m",
        "symbol": {
```

```
            "value": "m",
            "type": "http://labs.metoffice.gov.uk/edr/metadata/units/m"
          }
        },
        "observedProperty": {
          "id": "http://codes.wmo.int/common/quantity-kind/_horizontalVisibility",
          "label": "Visibility"
        },
        "measurementType": {
          "method": "instantaneous",
          "period": "PT0M"
        }
      },
      "Air Temperature": {
        "type": "Parameter",
        "description": "",
        "unit": {
          "label": "degC",
          "symbol": {
            "value": "\u00b0C",
            "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degC"
          }
        },
        "observedProperty": {
          "id": "http://codes.wmo.int/common/quantity-kind/_airTemperature",
          "label": "Air Temperature"
        },
        "measurementType": {
          "method": "instantaneous",
          "period": "PT0M"
        }
      },
      "Dew point": {
        "type": "Parameter",
        "description": "",
        "unit": {
          "label": "degC",
          "symbol": {
            "value": "\u00b0C",
            "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degC"
          }
        },
        "observedProperty": {
          "id": "http://codes.wmo.int/common/quantity-kind/_dewPointTemperature",
          "label": "Dew point"
        },
        "measurementType": {
          "method": "instantaneous",
          "period": "PT0M"
        }
      },
      "Runway Visibility": {
        "type": "Parameter",
        "description": "Runway Visibile Range",
        "unit": {
          "label": "m",
          "symbol": {
            "value": "m",
            "type": "http://labs.metoffice.gov.uk/edr/metadata/units/m"
          }
        },
        "observedProperty": {
          "id": "http://codes.wmo.int/common/quantity-kind/_runwayVisualRangeRvr",
          "label": "Runway Visibility"
        },
        "measurementType": {
          "method": "instantaneous",
          "period": "PT0M"
        }
      },
      "Weather": {
        "type": "Parameter",
        "description": "Aerodrome recent weather",
        "unit": {
          "label": "weather",
          "symbol": {
            "value": "",
            "type": "http://codes.wmo.int/49-2/AerodromeRecentWeather"
          }
        },
        "observedProperty": {
          "id": "http://codes.wmo.int/49-2/AerodromeRecentWeather",
          "label": "Weather"
        },
        "measurementType": {
          "method": "instantaneous",
          "period": "PT0M"
        }
      },
      "Sky condition": {
        "type": "Parameter",
        "description": "Sky condition",
        "unit": {
          "label": "sky",
```

```
            "symbol": {
              "value": "",
              "type": "http://{server}"
            }
          },
          "observedProperty": {
            "id": "",
            "label": "Sky condition"
          },
          "measurementType": {
            "method": "instantaneous",
            "period": "PT0M"
          }
        },
        "Mean Sea Level Pressure": {
          "type": "Parameter",
          "description": "",
          "unit": {
            "label": "hPa",
            "symbol": {
              "value": "hPa",
              "type": "http://labs.metoffice.gov.uk/edr/metadata/units/hPa"
            }
          },
          "observedProperty": {
            "id": "http://codes.wmo.int/bufr4/b/10/_051",
            "label": "Mean Sea Level Pressure"
          },
          "measurementType": {
            "method": "instantaneous",
            "period": "PT0M"
          }
        },
        "Station Level Pressure": {
          "type": "Parameter",
          "description": "",
          "unit": {
            "label": "hPa",
            "symbol": {
              "value": "hPa",
              "type": "http://labs.metoffice.gov.uk/edr/metadata/units/hPa"
            }
          },
          "observedProperty": {
            "id": "http://codes.wmo.int/bufr4/b/10/_004",
            "label": "Station Level Pressure"
          },
          "measurementType": {
            "method": "instantaneous",
            "period": "PT0M"
          }
        }
      }
    }
  ]
}
```


The **Collection** resource provides detailed information about the collection identified in a request. Some of the information returned includes the supported geographic extent, data queries, coordinate reference systems, output formats, and parameter names.

Below is an extract from the response to the request http://labs.metoffice.gov.uk/edr/collections/metar_demo?f=json


```
{
  "id": "metar_demo",
  "title": "Metar observations EDR demonstrator",
  "description": "API to access 24 hours of Global Metar Observation data (not for operational use)",
  "keywords": [
    "Metar observation",
    "ICAO identifier",
    "Wind Direction",
    "Wind Speed",
    "Wind Gust",
    "Visibility",
    "Air Temperature",
    "Dew point",
    "Runway Visibility",
    "Weather",
    "Sky condition",
    "Mean Sea Level Pressure",
    "Station Level Pressure",
    "description",
    "restrictions",
    "collection",
    "position",
    "radius",
    "area",
    "location"
  ],
  "links": [
```

```
    {
      "href": "http://labs.metoffice.gov.uk/collections/metar_demo",
      "hreflang": "en",
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "http://labs.metoffice.gov.uk/collections/metar_demo?f=html",
      "hreflang": "en",
      "rel": "alternate",
      "type": "text/html"
    },
    {
      "href": "http://labs.metoffice.gov.uk/collections/metar_demo?f=xml",
      "hreflang": "en",
      "rel": "alternate",
      "type": "application/xml"
    },
    {
      "href": "https://www.aviationweather.gov/metar/help",
      "hreflang": "en",
      "rel": "service-doc",
      "type": "text/html",
      "title": ""
    },
    {
      "href": "https://www.weather.gov/disclaimer",
      "hreflang": "en",
      "rel": "restrictions",
      "type": "text/html",
      "title": ""
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/position",
      "hreflang": "en",
      "rel": "data"
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/radius",
      "hreflang": "en",
      "rel": "data"
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/area",
      "hreflang": "en",
      "rel": "data"
    },
    {
      "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/locations",
      "hreflang": "en",
      "rel": "data"
    }
  ],
  "extent": {
    "spatial": {
      "bbox": [
        -180.0,
        -89.9,
        180.0,
        89.9
      ],
      "crs": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
    },
    "temporal": {
      "interval": [
        "R36/2021-10-03T03:00Z/PT1H"
      ],
      "trs": "TIMECRS[\"DateTime\",TDATUM[\"Gregorian Calendar\"],CS[TemporalDateTime,1],AXIS[\"Time (T)\",future]"
    }
  },
  "data_queries": {
    "position": {
      "link": {
        "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/position",
        "hreflang": "en",
        "rel": "data",
        "variables": {
          "title": "Position query",
          "query_type": "position",
          "output_formats": [
            "CoverageJSON",
            "GeoJSON",
            "IWXXM"
          ],
          "default_output_format": "GeoJSON",
          "crs_details": [
            {
              "crs": "CRS84",
              "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
            }
```

```
          ]
        }
      }
    },
    "radius": {
      "link": {
        "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/radius",
        "hreflang": "en",
        "rel": "data",
        "variables": {
          "title": "Radius query",
          "description": "Radius query",
          "query_type": "radius",
          "output_formats": [
            "CoverageJSON",
            "GeoJSON",
            "IWXXM"
          ],
          "default_output_format": "GeoJSON",
          "within_units": [
            "km",
            "miles"
          ],
          "crs_details": [
            {
              "crs": "CRS84",
              "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
            }
          ]
        }
      }
    },
    "area": {
      "link": {
        "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/area",
        "hreflang": "en",
        "rel": "data",
        "variables": {
          "title": "Area query",
          "query_type": "area",
          "output_formats": [
            "CoverageJSON",
            "GeoJSON",
            "IWXXM"
          ],
          "default_output_format": "CoverageJSON",
          "crs_details": [
            {
              "crs": "CRS84",
              "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
            }
          ]
        }
      }
    },
    "locations": {
      "link": {
        "href": "http://labs.metoffice.gov.uk/edr/collections/metar_demo/locations",
        "hreflang": "en",
        "rel": "data",
        "variables": {
          "title": "Location query",
          "description": "Location query",
          "query_type": "locations",
          "output_formats": [
            "CoverageJSON",
            "GeoJSON",
            "CSV"
          ],
          "default_output_format": "GeoJSON",
          "crs_details": [
            {
              "crs": "CRS84",
              "wkt": "GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",
\"6326\"]],PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",
\"4326\"]]"
            }
          ]
        }
      }
    }
  },
  "crs": [
    "CRS84"
  ],
  "output_formats": [
    "CoverageJSON",
    "GeoJSON",
    "IWXXM"
  ],
```

```
  "parameter_names": {
    "Metar observation": {
      "type": "Parameter",
      "description": "Source Metar observation",
      "unit": {
        "label": "",
        "symbol": {
          "value": "",
          "type": "http://codes.wmo.int/wmdr/DataFormat/FM-15-metar"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/wmdr/DataFormat/FM-15-metar",
        "label": "Metar observation"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "ICAO identifier": {
      "type": "Parameter",
      "description": "ICAO identifier",
      "unit": {
        "label": "",
        "symbol": {
          "value": "",
          "type": "https://en.wikipedia.org/wiki/ICAO_airport_code"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/bufr4/b/01/_063",
        "label": "ICAO identifier"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Wind Direction": {
      "type": "Parameter",
      "description": "Wind Direction",
      "unit": {
        "label": "degree true",
        "symbol": {
          "value": "\u00b0",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degree"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_aerodromeMeanWindDirection",
        "label": "Wind Direction"
      },
      "measurementType": {
        "method": "mean",
        "period": "-PT10M/PT0M"
      }
    },
    "Wind Speed": {
      "type": "Parameter",
      "description": "Wind Speed",
      "unit": {
        "label": "mph",
        "symbol": {
          "value": "mph",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/mph"
        }
      },
      "observedProperty": {
        "id": " http://codes.wmo.int/common/quantity-kind/aerodromeMeanWindSpeed",
        "label": "Wind Speed"
      },
      "measurementType": {
        "method": "mean",
        "period": "-PT10M/PT0M"
      }
    },
    "Wind Gust": {
      "type": "Parameter",
      "description": "Wind Gust",
      "unit": {
        "label": "mph",
        "symbol": {
          "value": "mph",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/mph"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_aerodromeMaximumWindGustSpeed",
        "label": "Wind Gust"
      },
      "measurementType": {
        "method": "maximum",
        "period": "-PT10M/PT0M"
```

```
      }
    },
    "Visibility": {
      "type": "Parameter",
      "description": "Visibility",
      "unit": {
        "label": "m",
        "symbol": {
          "value": "m",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/m"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_horizontalVisibility",
        "label": "Visibility"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Air Temperature": {
      "type": "Parameter",
      "description": "",
      "unit": {
        "label": "degC",
        "symbol": {
          "value": "\u00b0C",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degC"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_airTemperature",
        "label": "Air Temperature"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Dew point": {
      "type": "Parameter",
      "description": "",
      "unit": {
        "label": "degC",
        "symbol": {
          "value": "\u00b0C",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/degC"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_dewPointTemperature",
        "label": "Dew point"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Runway Visibility": {
      "type": "Parameter",
      "description": "Runway Visibile Range",
      "unit": {
        "label": "m",
        "symbol": {
          "value": "m",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/m"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/common/quantity-kind/_runwayVisualRangeRvr",
        "label": "Runway Visibility"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Weather": {
      "type": "Parameter",
      "description": "Aerodrome recent weather",
      "unit": {
        "label": "weather",
        "symbol": {
          "value": "",
          "type": "http://codes.wmo.int/49-2/AerodromeRecentWeather"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/49-2/AerodromeRecentWeather",
        "label": "Weather"
      },
      "measurementType": {
        "method": "instantaneous",
```

```
        "period": "PT0M"
      }
    },
    "Sky condition": {
      "type": "Parameter",
      "description": "Sky condition",
      "unit": {
        "label": "sky",
        "symbol": {
          "value": "",
          "type": "http://{server}"
        }
      },
      "observedProperty": {
        "id": "",
        "label": "Sky condition"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Mean Sea Level Pressure": {
      "type": "Parameter",
      "description": "",
      "unit": {
        "label": "hPa",
        "symbol": {
          "value": "hPa",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/hPa"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/bufr4/b/10/_051",
        "label": "Mean Sea Level Pressure"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    },
    "Station Level Pressure": {
      "type": "Parameter",
      "description": "",
      "unit": {
        "label": "hPa",
        "symbol": {
          "value": "hPa",
          "type": "http://labs.metoffice.gov.uk/edr/metadata/units/hPa"
        }
      },
      "observedProperty": {
        "id": "http://codes.wmo.int/bufr4/b/10/_004",
        "label": "Station Level Pressure"
      },
      "measurementType": {
        "method": "instantaneous",
        "period": "PT0M"
      }
    }
  }
}
```


Query Resources

Query resources are spatio-temporal queries which support operation of the API for the access and use of the spatio-temporal data resources.

Query resources share several common parameters, which makes it easier for developers to implement the queries.

Where the query applies to a collection, the pattern is as follows:

`/collections/{collectionId}/{queryType}`

The parameter `queryType` can be one of the following:

- position
- area
- cube
- trajectory
- corridor
- radius
- instances
- locations
- items

Where the query applies to an instance, the pattern is as follows:

```
/collections/{collectionId}/instances/{instanceId}/{queryType}
```

AREA QUERY RESOURCES OF OGC API - EDR

An area is a region specified with a geographic envelope that may have vertical dimension. An illustration, created using NASA WorldWind, is shown below.

The `area` query resource returns data for the defined area. The resource offers a convenience mechanism for querying the API by area, using a Well Known Text (WKT) POLYGON geometry.

The path to the resource is shown below:

```
/collections/{collectionId}/area
```

The paths accepts the following parameters:

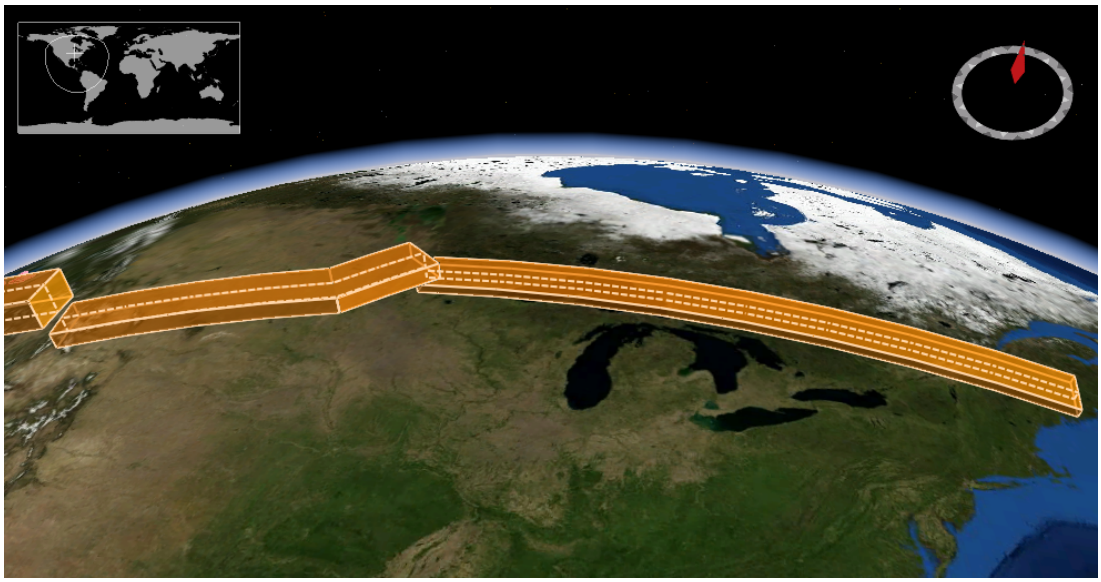
- coords
- z
- parameter-name
- datetime
- crs
- f

An example request is shown below.

```
http://example.org/edr/collections/gfs-pressure_at_height/area?
coords=POLYGON((-0.898132%2051.179362,-0.909119%2051.815488,0.552063%2051.818884,0.560303%2051.191414,-0.898132%2051.179362))&par
ameter-name=Pressure_height_above_ground&datetime=2022-01-19T06:00Z/2022-01-19T12:00Z&z=80/80&crs=CRS84&f=CoverageJSON
```

CORRIDOR QUERY RESOURCES OF OGC API - EDR

A corridor is a two parameter set of points around a trajectory. An illustration, created using NASA WorldWind, is shown below.



The `corridor` query resource returns data for the defined corridor. The resource offers a convenience mechanism for querying the API by corridor, using a Well Known Text (WKT) LINESTRING geometry, or alternatively subclasses LINESTRINGZ, LINESTRINGM, LINESTRINGZM.

The path to the resource is shown below:

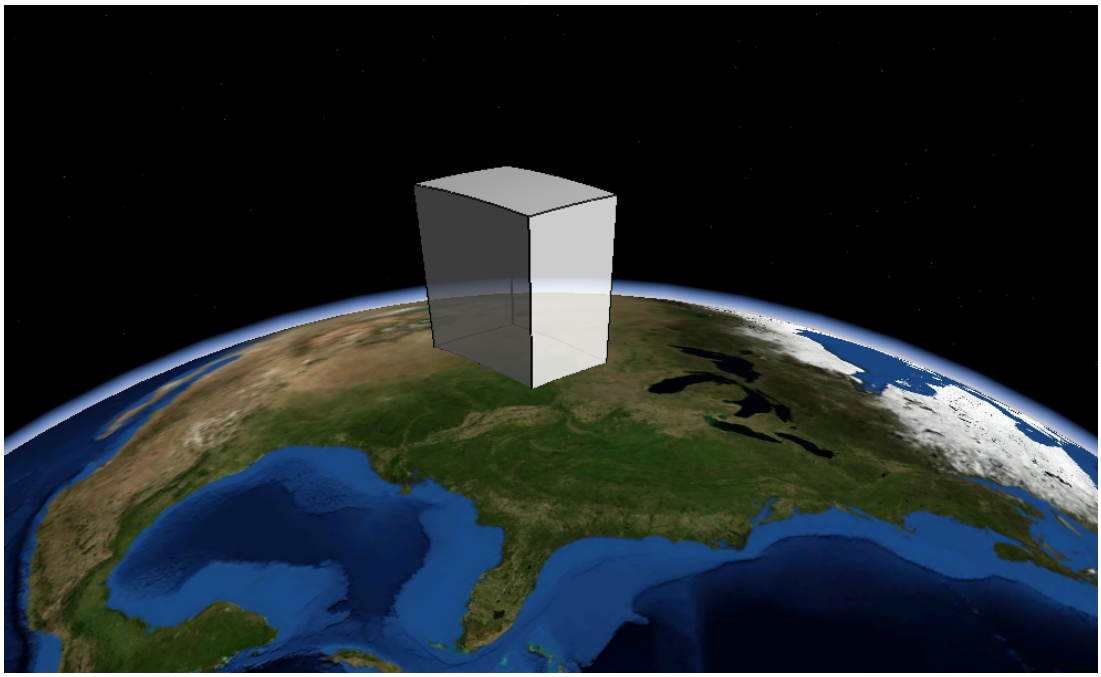
```
/collections/{collectionId}/corridor
```

The paths accepts the following parameters:

- coords
- corridor-width
- corridor-height
- width-units
- height-units
- z
- parameter-name
- datetime
- crs
- f

CUBE QUERY RESOURCES OF OGC API - EDR

A cube is a rectangular area, with a vertical extent. An illustration, created using NASA WorldWind, is shown below.



The `cube` query resource returns data for a defined cube. The resource offers a convenience mechanism for querying the API using a bounding box (BBOX) defining a cube.

The path to the resource is shown below:

```
/collections/{collectionId}/cube
```

The paths accepts the following parameters:

- bbox
- z
- parameter-name
- datetime
- crs
- f

INSTANCES QUERY RESOURCES OF OGC API - EDR

The `instances` query resource retrieves metadata about instances of a collection. The resource enables support for multiple instances or versions of the same underlying data source to be accessed by the API.

The path to the resource is shown below:

```
/collections/{collectionID}/instances/{instanceID}/{queryType}
```

ITEMS (FEATURES) QUERY RESOURCES OF OGC API - EDR

The `items` query resource offers an OGC API - Features endpoint that may be used to catalog pre-existing EDR sampling features.

Example use cases of this resource include:

- existence of a monitoring location
- cached query
- cataloguing of anomalies in a data

The path to the resource is shown below:

```
/collections/{collectionId}/items
```

An example request is below.

`http://example.org/edr/collections/mocov-daily_global/items`

LOCATIONS QUERY RESOURCES OF OGC API - EDR

The `locations` query resource returns a list of location identifiers and relevant metadata for the collection.

The location identifier can be anything as long as it is unique for the required position (e.g. a GeoHash).

The path to the resource is shown below:

`/collections/{collectionId}/locations`

An example request is below.

`http://example.org/edr/collections/obs_demo/locations`

POSITION QUERY RESOURCES OF OGC API - EDR

A position is a data type that describes a point or geometry potentially occupied by an object or person. An illustration, created using NASA WorldWind, is shown below.

The `position` query resource returns data for the requested position. The resource offers a convenience mechanism for querying the API using a Well Known Text (WKT) POINT geometry defining a position.

The path to the resource is shown below:

`/collections/{collectionId}/position`

The paths accepts the following parameters:

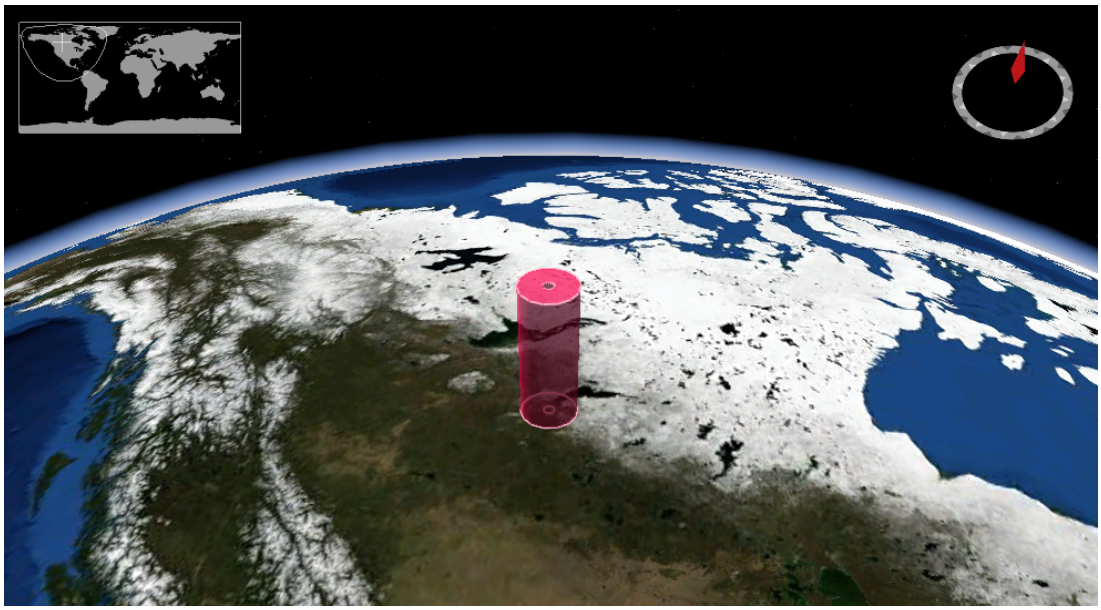
- coords
- z
- parameter-name
- datetime
- crs
- f

An example request is shown below.

```
http://example.org/edr/collections/obs_demo/position?coords=POINT(0.00577%2051.562608)&parameter-name=Wind%20Direction&datetime=2022-01-19T10:00Z/2022-01-19T12:00Z&crs=CRS84&f=GeoJSON
```

RADIUS QUERY RESOURCES OF OGC API - EDR

A radius is a region specified with a geographic position and radial distance. An illustration, created using NASA WorldWind, is shown below.



The `radius` query resource returns data for a defined radius. The resource offers a convenience mechanism for querying the API by radius.

The path to the resource is shown below:

```
/collections/{collectionId}/radius
```

The paths accepts the following parameters:

- coords
- within
- width-units
- z
- parameter-name
- datetime
- crs
- f

An example request is shown below.

```
http://example.org/edr/collections/obs_demo/radius?coords=POINT(-0.095882%2051.512983)&within=50&within-units=km&parameter-name=Wind%20Direction&datetime=2022-01-19T04:00Z/2022-01-19T06:00Z&crs=CRS84&f=GeoJSON
```

TRAJECTORY QUERY RESOURCES OF OGC API - EDR

A trajectory is a path of a moving point described by a one parameter set of points. An illustration, created using NASA WorldWind, is shown below.

The `trajectory` query resource returns data for the defined trajectory. The resource offers a convenience mechanism for querying the API by trajectory, using a Well Known Text (WKT) LINESTRING geometry, or alternatively the specializations LINESTRINGZ, LINESTRINGM, LINESTRINGZM.

The path to the resource is shown below:

```
/collections/{collectionId}/trajectory
```

The paths accepts the following parameters:

- coords
- z
- parameter-name
- datetime
- crs
- f

An example request is shown below.

```
http://example.org/edr/collections/gfs-pressure_at_height/trajectory?coords=LINESTRING(-3.56 53.695,-3.546 53.696,-3.532 53.697)&parameter-name=Height&crs=CRS84&f=CoverageJSON
```

10.10.3 Summary

OGC API - Environmental Data Retrieval provides a family of lightweight interfaces to access Environmental Data resources. Each resource addressed by an EDR API maps to a defined query pattern. In this deep dive, we provided an overview of the standard and described each of these query patterns in detail.

10.11 OGC SensorThings API

Audience

Students that are familiar with web services and want to have an overview of the SensorThings Application Programming Interface (API) standard

Learning Objectives

At completion of the module students will be able to:

- Explain what the SensorThings API is
- Describe what can be done with the SensorThings API
- Understand how to retrieve data through the SensorThings API
- Understand how to publish sensor-collected data through the SensorThings API
- Be able to find a SensorThings API endpoint and use it through a client

10.11.1 Introduction

The Internet of Things (IoT) is a global information infrastructure that enables advanced services by interconnecting both physical and virtual "things" based on existing and evolving interoperable information and communication technologies [ITU-T]. To facilitate geospatial interoperability between devices in the IoT, the OGC has published the OGC SensorThings API.

The OGC SensorThings API is a multi-part standard for an open and geospatial-enabled approach for interconnecting devices, data, and applications of the Internet of Things (IoT). The first part of the standard describes the interface for Sensing. The second part describes the interface for Tasking. The Sensing part standardizes the management and retrieval of observations and metadata from heterogeneous IoT sensor systems. The Tasking part provides a standard way for parameterizing - also called tasking - of IoT devices that can be instructed to carry out observations or perform other functions. SensorThings also includes an extension, STAplus, specifically developed to address the requirements from the Citizen Science community.

Note

This tutorial module is not intended to be a replacement to the actual **OGC SensorThings API Part 1: Sensing** standard. The tutorial intentionally focuses on a subset of capabilities in order to get the student started with using the standard. Please refer to the **OGC SensorThings API Part 1: Sensing** standard for additional detail.

Background

History

The OGC SensorThings API is based on the existing OGC Sensor Web Enablement (SWE) standards. It was developed to address the specific needs of the IoT community. SensorThings API Part 1: Sensing version 1.0 was approved by the OGC Technical Committee in February 2016.

Versions

OGC SensorThings API Part 1: Sensing Version 1.1 and **OGC SensorThings API Part 2 – Tasking Core** Version 1.0.0 are the current latest versions. The current latest version of the **STAplus extension** is 1.0.0.

Test suite

A test suite is available for:

- SensorThings API - Part 1

Public Endpoints

A list of public endpoints can be found here: https://github.com/opengeospatial/sensorthings/blob/master/PublicEndPoints.md

USAGE

The SensorThings API allows for the access and dissemination of sensor-collected data about any object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication

networks. The data is accessed through a resource-centric interface that is based on Representational state transfer (REST) principles. The data returned by the API is serialized in JavaScript Object Notation (JSON).

The benefit of adopting REST and JSON for the SensorThings API is that they offer greater efficiency in devices of constrained Size, Weight and Power (SWaP) such as microcomputers, smart home controllers, nano-Unmanned Aerial Vehicles (UAVs), smartphones, smart watches and tablets. The use of REST also makes it easier for web developers and the applications they implement to access data through resource-centric Uniform Resource Locator (URL) patterns.

- **OGC SensorThings API Part 1: Sensing** - provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems.
- **OGC SensorThings API Part 2: Tasking Core** - provides a standard way for parameterizing - also called tasking - of taskable IoT devices, such as individual sensors and actuators, composite consumer / commercial / industrial / smart cities in-situ platforms, mobile and wearable devices, or even unmanned systems platforms such as drones, satellites, connected and autonomous vehicles, etc.
- **OGC SensorThings API Extension: STAplus** - is designed to support a model in which observations are owned by different users. This results in requirements for the ownership concept. In addition to the ownership, users may express a license for ensuring proper re-use of their observations. The STAplus extension also supports expressing explicit relations between observations as well as between observations and external resources. Relations can enrich observations to enable future extensions supporting Linked Data, RDF and SPARQL. Observation group(s) allow the grouping of observations that belong together.

Note

The rest of this tutorial will focus on Version 1.0 of the Part 1 of the standard (e.g.: Sensing). Version 1.1 of SensorThings API Part 1 is an update to version 1.0 that is (mostly) backwards compatible with version 1.0.

RELATION TO OTHER OGC STANDARDS

- Sensor Observation Service Interface Standard (SOS): The SensorThings API is designed specifically to enable the dissemination of observations from resource-constrained IoT devices and the Web developer community. In contrast to SOS, the SensorThings API uses approaches that are considered more efficient for example, REST, JSON and the Message Queuing Telemetry Transport (MQTT).
- Web Feature Service Interface Standard (WFS) : The WFS standard is designed to allow for serving feature types of any kind. Other than requiring the data to be serializable in Geography Markup Language (GML), WFS does not place any other significant constraints. In contrast, SensorThings API formalized how specific entities and concepts should be represented and serialized.

Overview of Resources

SensorThings API provides a serious of entities as resources. The following is a list entities supported by the API:

Thing

```
The OGC SensorThings API follows the ITU-T definition, i.e., with
regard to the Internet of Things, a thing is an object of the
physical world (physical things) or the information world (virtual
```

```
things) that is capable of being identified and integrated into
communication networks ITU-T.
```

Location

```
The Location entity locates the Thing or the Things it associated
with. A Thing's Location entity is defined as the last known
location of the Thing.
```

HistoricalLocation

```
A Thing's HistoricalLocation entity set provides the times of the
current (i.e., last known) and previous locations of the Thing.
```

Datastream

```
A Datastream groups a collection of Observations measuring the same
ObservedProperty and produced by the same Sensor.
```

Sensor

```
A Sensor is an instrument that observes a property or phenomenon
with the goal of producing an estimate of the value of the property.
```

ObservedProperty

```
An ObservedProperty specifies the phenomenon of an Observation.
```

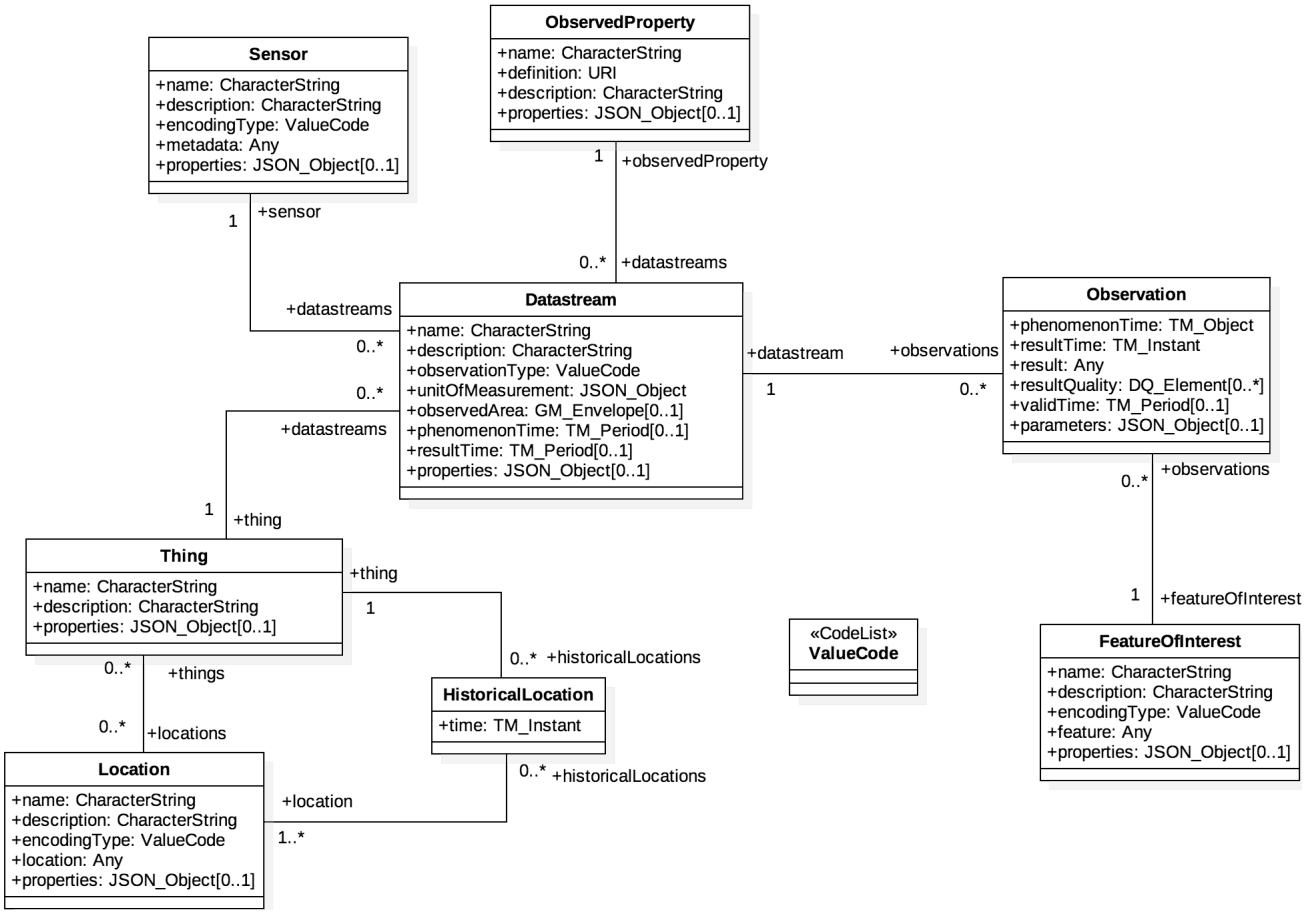
Observation

```
An Observation is the act of measuring or otherwise determining the
value of a property.
```

FeatureOfInterest

```
The phenomenon against which an observation is made is a property of
the feature of interest.
```

The figure below shows the relations between sensing entities.



Example

This SensorThings API server publishes sample data about available bikes and docks from a Toronto bike share station.

An example request to retrieve sensors through the API is shown below.

http://toronto-bike-snapshot.sensorup.com/v1.0/Sensors

The response, which is presented below, reports that there are two sensors: one for tracking how many docks are available in a bike station and another sensor for tracking how many bikes are available in a bike station.


```
{"@iot.count":2,
    "value":[
        {"@iot.id":4,"@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Sensors(4)","description": "A sensor for tracking how many docks are
available in a bike station","name": "available_docks","encodingType": "text/plan","metadata": "https://member.bikesharetoronto.com/
stations","Datastreams@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Sensors(4)/Datastreams"
            },
        {"@iot.id":3,"@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Sensors(3)","description": "A sensor for tracking how many bikes are
available in a bike station","name": "available_bikes","encodingType": "text/plan","metadata": "https://member.bikesharetoronto.com/
stations","Datastreams@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Sensors(3)/Datastreams"
            }
        ]
}
```


The data returned by the service can be rendered by a desktop Geographic Information System (GIS) or a web application. Alternatively, it can be forwarded to an OGC API - Processes service for further processing.

Client usage

A client needs to know the location of the SensorThings API service to be able to interact with the server. The location is usually called the `endpoint` of the service and is represented by the service root URI. Resources available through the service can be accessed by appending a resource path and, optionally query options.

For example, the first line of the following URL is the service root URI. The second line is the resource path. The third line is the query option.

```
http://toronto-bike-snapshot.sensorup.com/v1.0
/Datastreams(206051)/Observations(1593917)
?$select=result
```

The link to the request is: http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(206051)/Observations(1593917)?$select=result

Checkout various available public endpoints here

10.11.2 Operations

The entities offered by a SensorThings API service can be accessed by appending a resource path to the service root URI. An example of a URL that retrieves observations is shown below.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations

An extract of the response is presented below. Notice how the instances of the requested entity are presented in a JSON array.


```
{"@iot.count":1594349,
    "@iot.nextLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$top=100&$skip=100","value":
        [
            {"@iot.id":1595550,"@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595550)","phenomenonTime":
"2017-02-16T21:55:12.841Z","result": "7","resultTime":null,"Datastream@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/
Observations(1595550)/Datastream","FeatureOfInterest@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595550)/
FeatureOfInterest"
                },
            {"@iot.id":1595551,"@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595551)","phenomenonTime":
"2017-02-16T21:55:12.841Z","result": "4","resultTime":null,"Datastream@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/
Observations(1595551)/Datastream","FeatureOfInterest@iot.navigationLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595551)/
FeatureOfInterest"
                },
                ...
        ]
}
```


Other entities can also be retrieved through resource paths of a similar pattern. The following table lists the resource paths of each entity type.

Entity Sets Offered

Entity Set	Method	Resource Path
Things	GET	/Things
Locations	GET	/Locations
Historical locations	GET	/HistoricalLocations
Datastreams	GET	/Datastreams
Sensors	GET	/Sensors
Observed properties	GET	/ObservedProperties
Observations	GET	/Observations
Features of interest	GET	/FeaturesOfInterest

In addition to accessing an entity, the property of an entity can also be accessed in a similar way by appending the name of the property to the resource path. The following is an example of a request that retrieves a property named `result` from a specific observation.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595550)/result

Examples of resource paths of properties are shown in the following table.

Property Resource Path Examples

Property	Method	Resource Path
Result of an observation with an ID of 1595550	GET	/Observations(1595550)/result
The name of a feature of interest	GET	/Sensor(4)/metadata
Coordinates of the feature observed by observation 1595550	GET	/Observations(1595550)/FeatureOfInterest/feature

Retrieval Options

$FILTER

The `$filter` system option allows clients to filter a collection of entities that are addressed by a request URL.

For example, the following request returns all Observations whose result is less than 15.00.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$filter=result%20lt%2015.00

$COUNT

The `$count` query option specifies whether the total count of items within a collection matching the request should be returned along with the result.

For example, the following request returns the total number of Observations in the collection, as well as the results. Changing the value of the `$count` option to false causes the count to be omitted from the response.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$count=true

$ORDERBY

The `$orderby` query option specifies the order in which items are returned from the service.

For example, the following request all Observations arranged in ascending order of the result property

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$orderby=result

$SKIP

The `$skip` query option specifies the number for the items of the queried collection that should be excluded from the result.

For example, the following request all Observations starting with the twenty-first Observation entity.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$skip=20

$TOP

The `$top` query option specifies the limit on the number of items returned from a collection of entities.

For example, the following request returns only the first six entities in the Observations collection.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$top=6

$EXPAND

The `$expand` query option enables the client to specify the set of properties to be included in a response by indicating that the related entities are to be represented inline.

For example, the following request returns the complete entity set of Things and their associated Datastreams.

http://toronto-bike-snapshot.sensorup.com/v1.0/Things?$expand=Datastreams

$SELECT

The `$select` query option enables the client to specify the set of properties to be included in a response by instructing the service to return only the properties explicitly requested.

For example, the following request returns each Observation entity with only the result and phenomenonTime properties listed.

http://toronto-bike-snapshot.sensorup.com/v1.0/Observations?$select=result,phenomenonTime

10.11.3 Demo

On this section, we explore different ways to access a SensorThings API server on:

http://toronto-bike-snapshot.sensorup.com/v1.0/

Web Client

We start exploring the different endpoints available in the server using a web browser. In alternative you could also use postman or curl.

RETURN BASE RESOURCE PATH

http://toronto-bike-snapshot.sensorup.com/v1.0/


```
{
  "value":[
      {
          "name": "Things",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/Things"
      },
      {
          "name": "Locations",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/Locations"
      },
      {
          "name": "HistoricalLocations",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/HistoricalLocations"
      },
      {
          "name": "Datastreams",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/Datastreams"
      },
      {
          "name": "Sensors",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/Sensors"
      },
      {
          "name": "Observations",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/Observations"
      },
      {
          "name": "ObservedProperties",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/ObservedProperties"
      },
      {
          "name": "FeaturesOfInterest",
          "url": "http://pm25-march.singapore2017.sensorup.com/v1.0/FeaturesOfInterest"
      }
      ]
      }
```


WHICH *THINGS* ARE AVAILABLE IN THE SERVER?

http://toronto-bike-snapshot.sensorup.com/v1.0/Things


```
{
"@iot.count":199,
"@iot.nextLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/
      Things?$top=100&$skip=100",
"value":[
 {
   "@iot.id":206047,
   "@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Things(206047)",
   "description": "Bloor St / Brunswick Ave Toronto bike share station with data
            of available bikes and available docks",
   "name": "7061:Bloor St / Brunswick Ave",
   "properties":{

   },
   ...
```

GETTING A *DATASTREAM* FOR A THING

http://toronto-bike-snapshot.sensorup.com/v1.0/Things(206047)/Datastreams


```
{
 "@iot.count":2,
 "value":[
 {
   "@iot.id":206051,
   "@iot.selfLink":
       "http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(206051)",
   "description":
       "... available docks count for the Toronto bike share station Bloor St ",
   "name": "7061:Bloor St / Brunswick Ave:available_docks",
   "observationType":
       "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation",
   "unitOfMeasurement":{
     "symbol": "{TOT}",
     "name": "dock count",
     "definition": "http://unitsofmeasure.org/ucum.html#para-50"
   },
   ....
```


Note

Datastreams define the unit of measurement


```
 "observationType":
     "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation",
 "unitOfMeasurement":{
   "symbol": "{TOT}",
   "name": "dock count",
   "definition": "http://unitsofmeasure.org/ucum.html#para-50"
 },
```


GETTING THE *OBSERVATIONS* RELATED TO A STREAM

http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(206051)/Observations


```
{
"@iot.count":3511,
"@iot.nextLink":
      "http://toronto-bike-snapshot.sensorup.com/...",
"value":[
 {
   "@iot.id":1595467,
   "@iot.selfLink":
      "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595467)",
   "phenomenonTime": "2017-02-16T21:55:12.233Z",
   "result": "23",
   "resultTime":null,
   "Datastream@iot.navigationLink":
      "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595467)/Datastream",
   "FeatureOfInterest@iot.navigationLink":
      "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595467)/FeatureOfInterest"
 },
```


COMPLEX QUERY

- Expands Datastreams and observations in one query
- Feature of Interest = 7000:Ft. York / Capreol Crt.
- Start time = 2017-01-01T11:30:00.000Z
- End time = 2017-03-01T11:30:00.000Z

Link

```
http://toronto-bike-snapshot.sensorup.com/v1.0/Things?
$expand=Datastreams/Observations/FeatureOfInterest&
$filter=Datastreams/Observations/FeatureOfInterest/
name eq '7000:Ft. York / Capreol Crt.' and
Datastreams/Observations/phenomenonTime ge 2017-01-01T11:30:00.000Z
and
Datastreams/Observations/phenomenonTime le 2017-03-01T11:30:00.000Z
```

COMPLEX QUERY RESPONSE

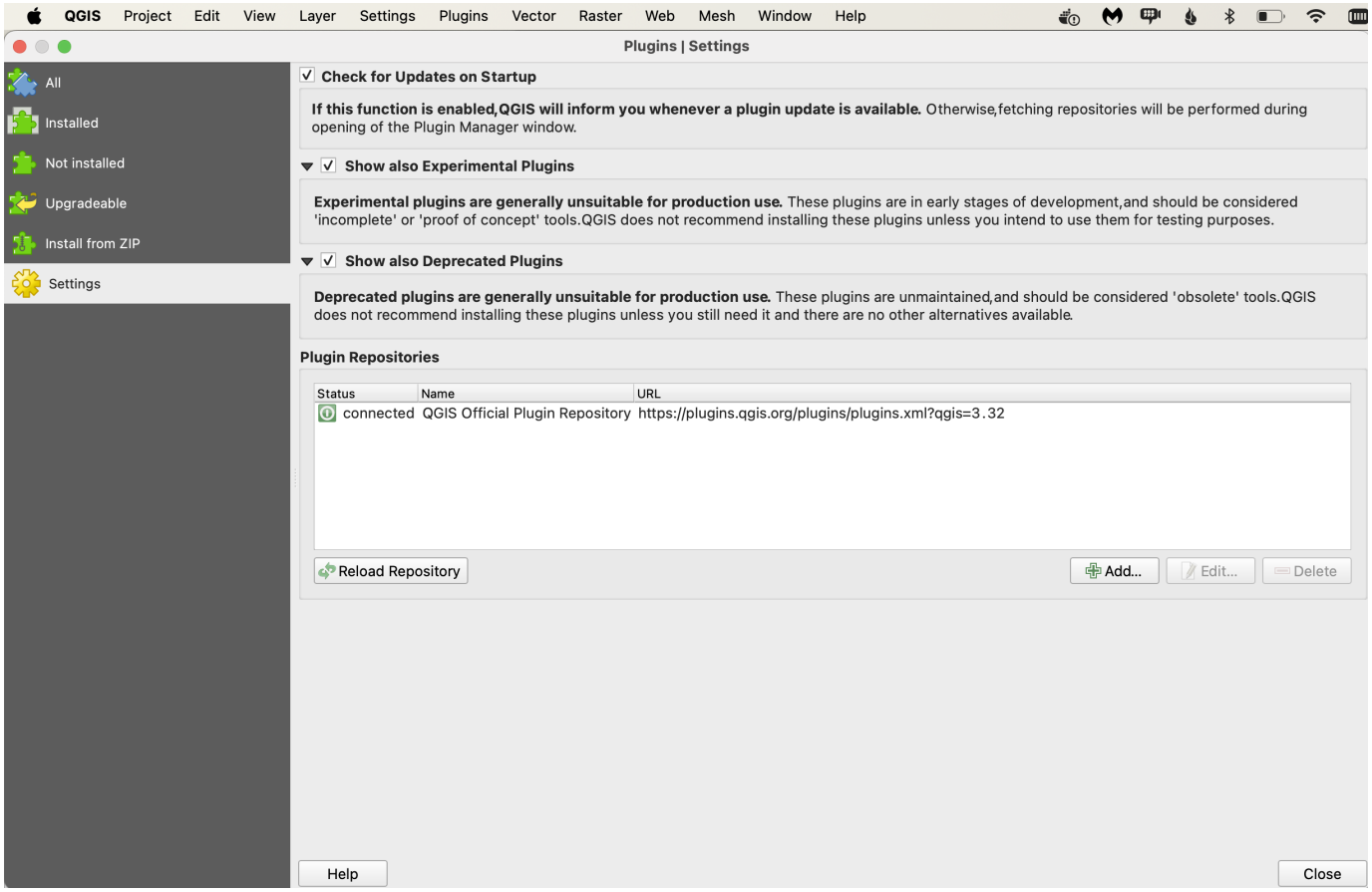

```
{
"@iot.count":1,
"value":[
 {
   "@iot.id":5,
   "@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Things(5)",
   "description":
         "Ft. York / Capreol Crt. Toronto bike share station available bikes and docks",
   "name": "7000:Ft. York / Capreol Crt.",
   "properties":{

   },
   "Datastreams":[
     {
       "@iot.id":9,
       "@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(9)",
       "description":
            "...available docks count for the Toronto bike share station Ft. York / Capreol Crt.",
       "name": "7000:Ft. York / Capreol Crt.:available_docks",
       "observationType":
            "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation",
       "unitOfMeasurement":{
         "symbol": "{TOT}",
         "name": "dock count",
         "definition": "http://unitsofmeasure.org/ucum.html#para-50"
       },
"Observations@iot.nextLink":
         ".../v1.0/Datastreams(9)/Observations?$top=100&$skip=100",
 "Observations":[
   {
     "@iot.id":1595545,
     "@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/Observations(1595545)",
     "phenomenonTime": "2017-02-16T21:55:12.797Z",
     "result": "10",
     "resultTime":null,
     "Datastream@iot.navigationLink":
           ".... /v1.0/Observations(1595545)/Datastream",
     "FeatureOfInterest":{
       "@iot.id":10,
       "@iot.selfLink": "http://toronto-bike-snapshot.sensorup.com/v1.0/FeaturesOfInterest(10)",
       "description": "  ...
```


QGIS Plugin

The SensorThings API plugin enables QGIS software to access dynamic data from sensors, using SensorThings API protocol.

In order to install this plugin from the QGIS repository, you will need to enable experimental plugins, in the plugins settings.

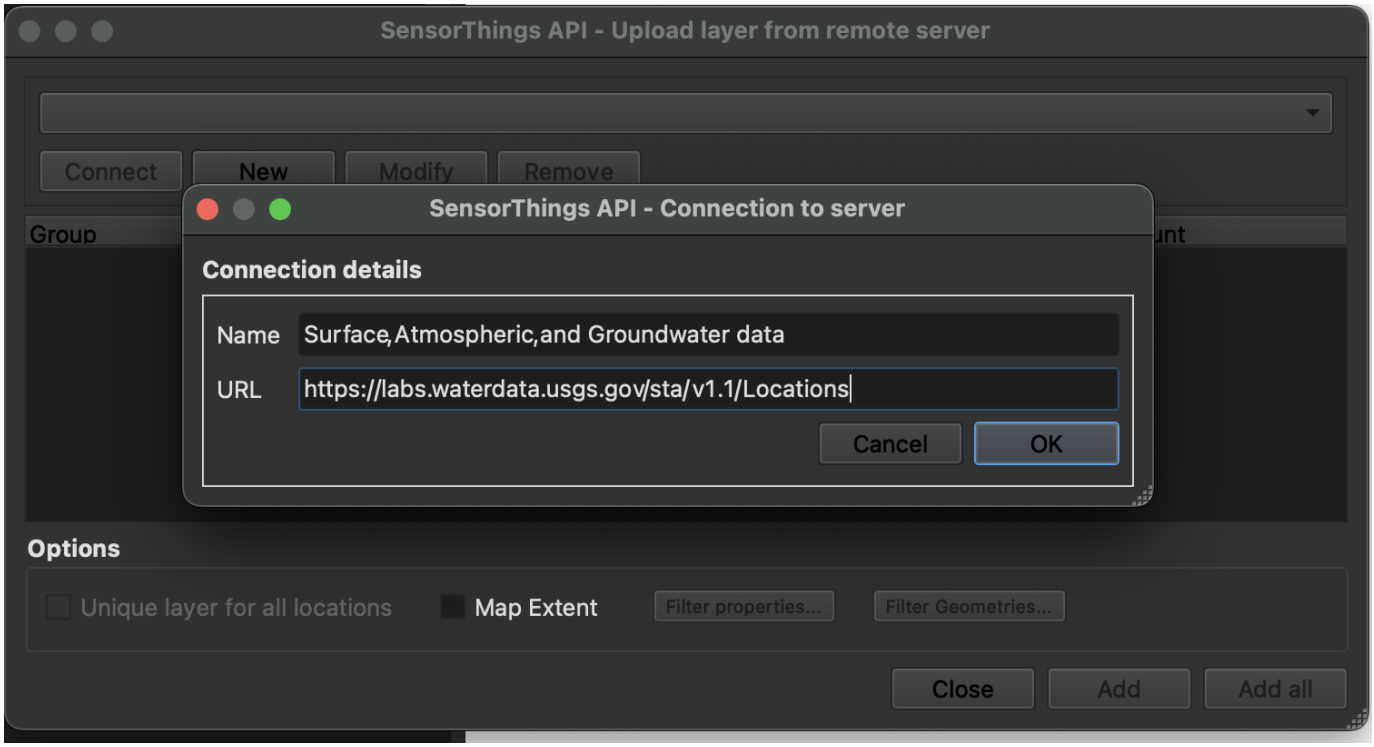


Open the plugin and enter SensorThings API with `/Locations`

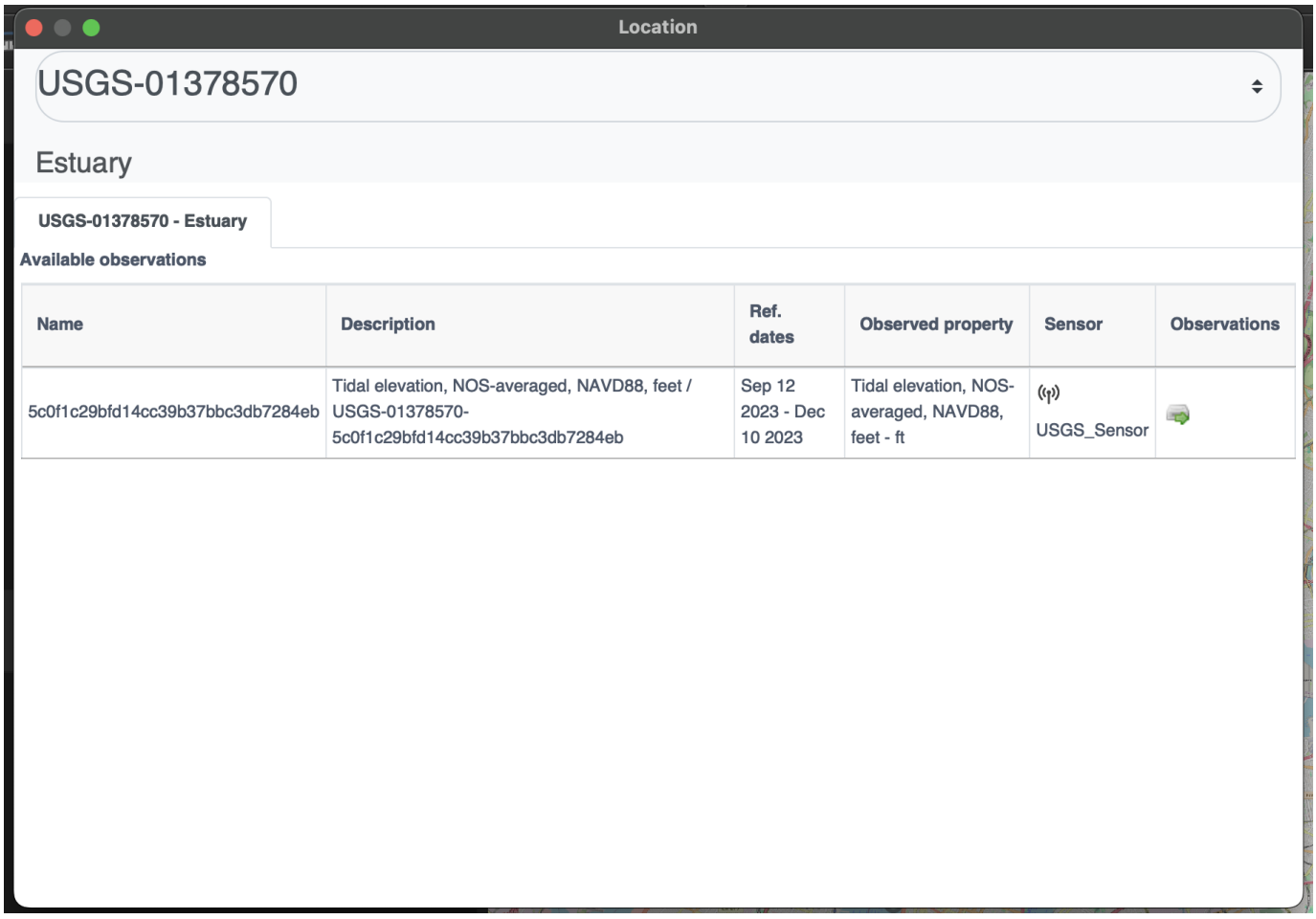
In our case we'll connect to

```
Name - Surface, Atmospheric, and Groundwater data
URL - https://labs.waterdata.usgs.gov/sta/v1.1/Locations
```

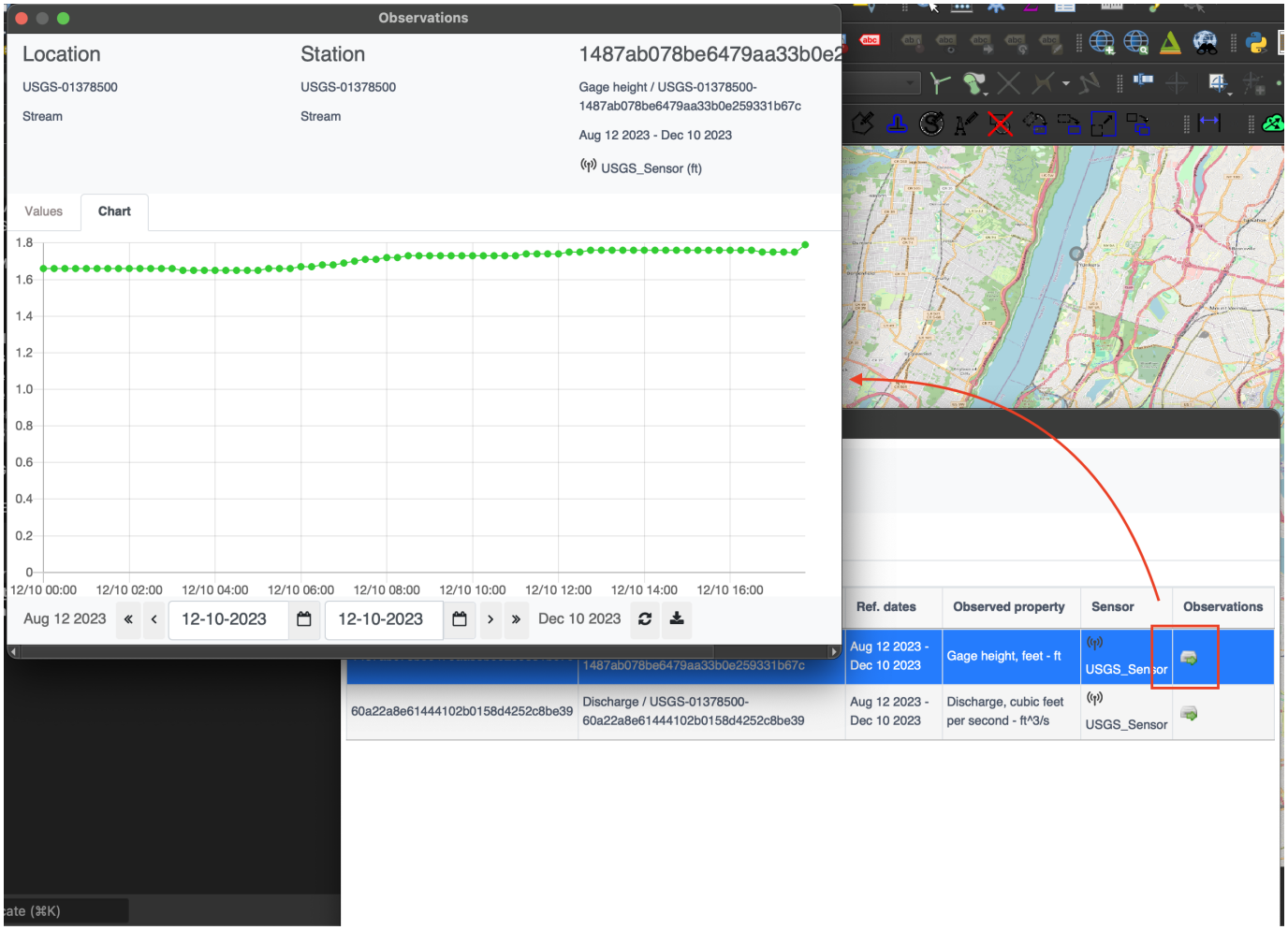
Now you can either add each sensors as new layer or combine all in one layer



Now we can check more information about each Location by activate `Show Location Information` and then clicking on sensor



Each sensor also has observation panel which allows us to see complete spatio-temporal data for each sensor in table and graph format



10.11.4 References

ITU-T, Overview of the Internet of things

SensorUp SensorThings API

10.11.5 Summary

The OGC SensorThings API provides an open and unified way to interconnect IoT devices, data, and applications over the Web. It builds on Web protocols and the OGC Sensor Web Enablement standards, and applies an easy-to-use REST-like style. This deep dive provided an overview of the entities and main operations made available by this standard.

11. OGC API Roadmap

The OGC API Roadmap highlights the current and planned standards efforts as well as related extensions to those standards. The Roadmap is useful for planning new functionality to users, as well as software implementation.

The current Roadmap can be found on the OGC Productboard as follows:

- Status
- Timeline
- Use Case

12. Security and OGC APIs

OGC APIs are designed using modern technologies in order to lower the barrier to geospatial data, services, and processes.

12.1 SSL/TLS

OGC APIs can be deployed using HTTP or HTTPS. It is strongly recommended to deploy any services using HTTPS so that clients can validate and verify authenticity of your services accordingly. Depending on how your system is architected, this may mean applying Secure Sockets Layer/Transport Layer Security (SSL/TLS) on your service host, or if you have a multi-layered deployment architecture, applying as part of your front-end services, at which point internal/inner communication may or may not be implemented using HTTP.

12.2 Access control

Open Standards and APIs are not only for Open Data. Implementing access control (authentication, authorization) is a critical component of many infrastructures and systems in order to maintain data integrity, authority and trust. Examples of requiring access control in OGC APIs includes (but is not limited to):

- securing all endpoints
- securing only specific endpoints
- allowing insert/update/delete capabilities on items in a collection
- allowing insert/update/delete capabilities on collections

Given that access control concerns, implementations and architectures exist for many domains, it is best to leverage industry standards for implementation. Given OGC API standards leverage the OpenAPI specification for service descriptions, one can use the OpenAPI Security Scheme Object to describe (not implement!) the access control mechanism(s) for the entire API as well as for a specific path/operation of the API.

Supported OpenAPI security schemes include:

- API key (`apiKey`)
- HTTP authentication (`http`)
- OAuth2 common flows (`oauth2`)
- OpenID Connect Discovery (`openIdConnect`)

Access control using HTTP Basic authentication:

```
"security": {
  "default": {
    "type": "http",
    "scheme": "basic",
    "description": "Please contact us for access information"
  }
}
```

Access control using an API key:

```
"security": {
  "default": {
    "type": "apiKey",
    "name": "api-key",
    "in": "query",
    "description": "Please see https://example.org/contact-us for more information"
  }
}
```

Access control using OAuth2:

```
"security": {
  "default": {
```

```
    "type": "oauth2",
    "authorizationUrl": "https://example.org/oauth/authorize",
    "flow": "implicit",
    "description": "Please see https://example.org/contact-us for more information"
    "scopes": {
        "read:roads": "read roads collection",
        "write:roads": "modify roads in the roads collection"
  }
}
```

Note

Implementing the above assumes that the required access control mechanisms are in place.

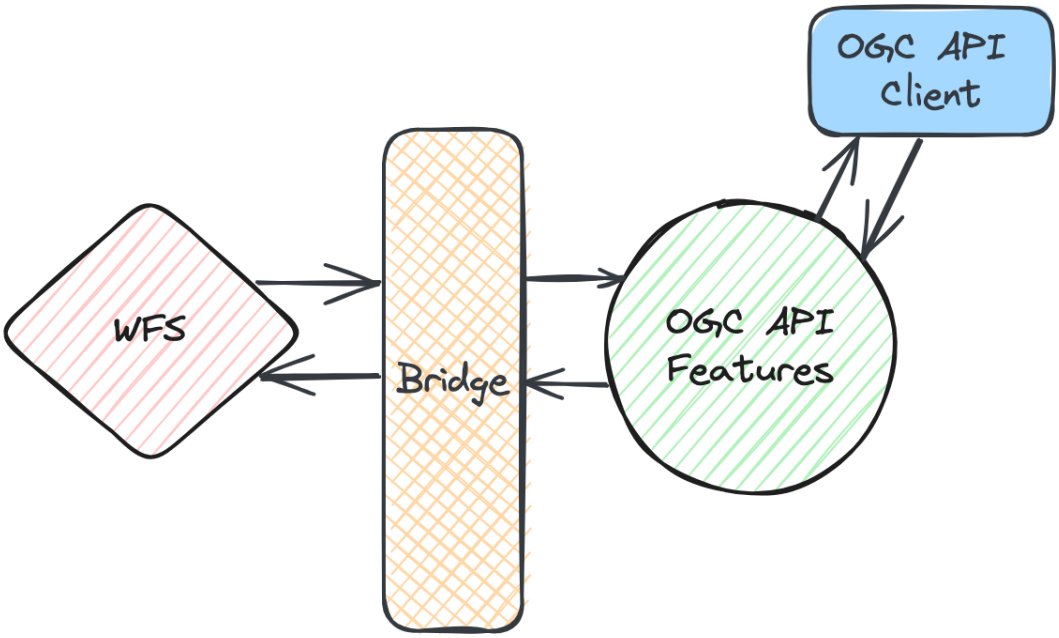
13. Transition and migration

According to the OGC Technical Committee Policies and Procedures, if an existing Standard is replaced in part or whole by one or more new Standards, then a special case of deprecation may occur resulting in the original Standard being labeled a "Legacy Standard."

This could be the case of many OWS standards, as they are replaced by more modern and more complete OGC APIs.

As with deprecated Standards, Legacy Standards are no longer supported, but they remain on the OGC website with a notification that the capabilities of the Standard have been replaced in whole or part by new Standard(s). The notification will clearly indicate that the Legacy Standard is not invalid, but that new implementations of the capabilities of the Standard are better served by the identified new Standard(s).

In the Diving into pygeoapi Workshop, you can find a section about the use of bridges to facilitate the migration from OWS to OGC API.

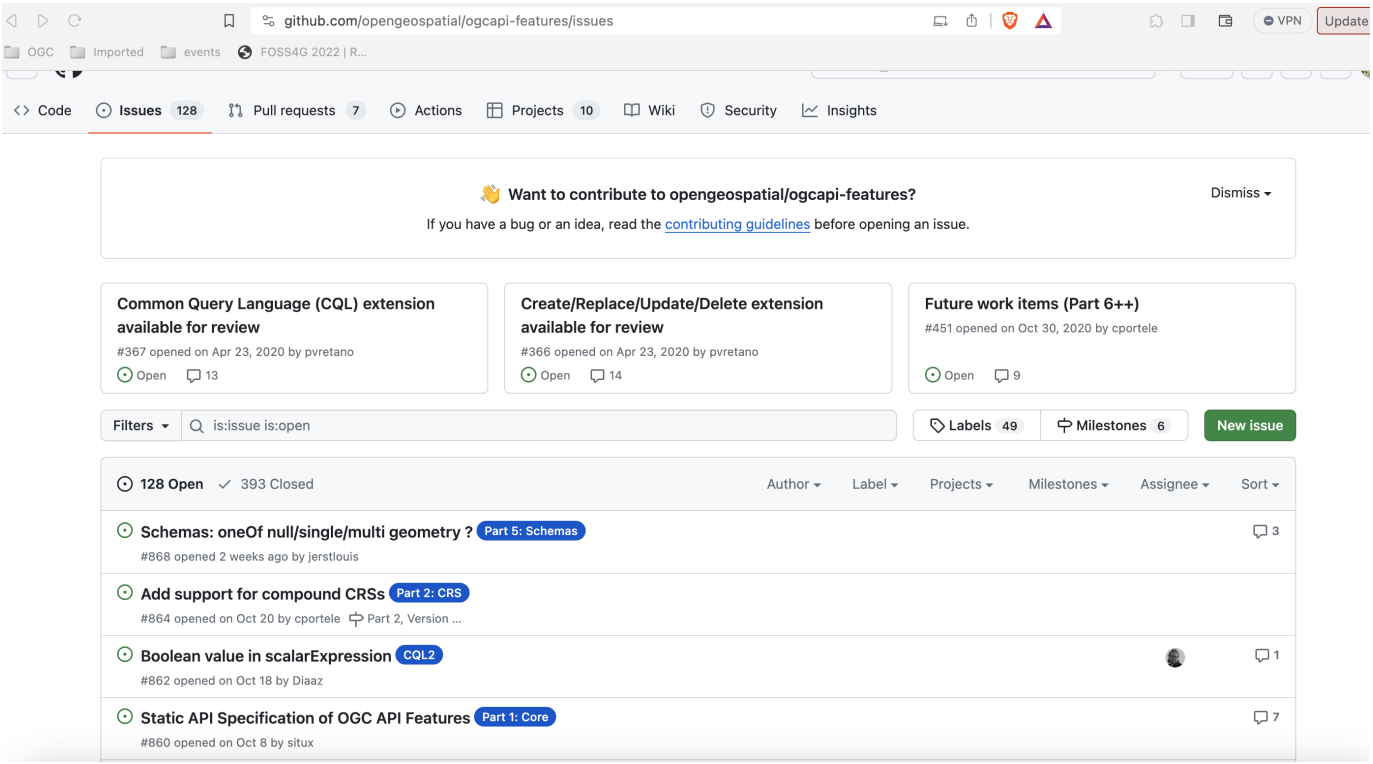


14. Getting involved

There are numerous ways to get involved with the development of OGC APIs. If you want to attend the Standards Working Group (SWG) meetings, and take part of the most formal component of standards development (including voting), an OGC membership is required. However, there are other opportunities to get involved that do not require an OGC membership. We present some of those below.

14.1 Specifications / GitHub

Most of the OGC API development takes place on public GitHub repositories. This means anyone can follow the development of the standards, from their early stages and even contribute through the usual mechanisms (e.g.: issue tracker, pull requests)



14.2 OGC Code sprints

The OGC Code sprints are an important part of the standards development process, as they provide a feedback loop from the developer community. During these events, working group members and developers join forces towards the common goal of pushing the standards development forward. This is done through discussions and prototyping. These are some characteristics of code sprints:

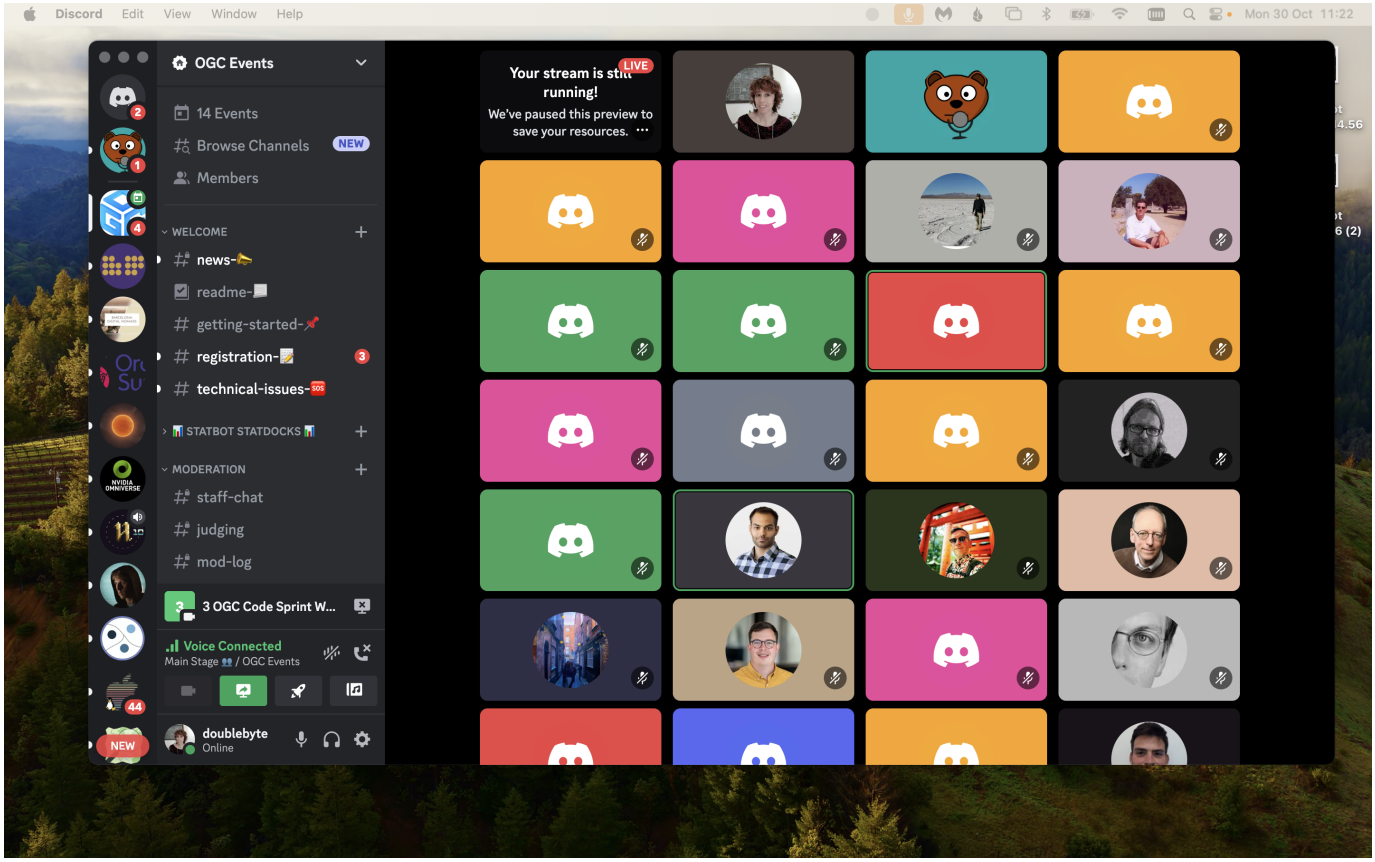
- Held regularly (around three times a year)
- Three-day collaborative, virtual/hybrid, events
- Inclusive to all OGC standards
- Often co-organised with alliance partners (e.g.: ISO, OSGeo, ASF)
- Feature developers from across the globe
- Feature a mentor stream, to onboard newcomers

Not *only* code!

Although most of the participants spend time coding (that is "why" it is called a code sprint, after all), we also welcome no-code activities, such as working on documentation, testing or GitHub issues.

14.3 OGC Events Discord server

The OGC Events Discord server provides a platform for running the code sprints. Its channel structure, facilitates focused discussions and we leverage the audio channels to run the code sprint meetings. In between code sprints, the Discord server also provides a meeting place to discuss topics related to OGC standards development and implementations.



14.4 Developer resources

You can learn more about the OGC API (and other OGC standards), from a developer-centric perspective on the OGC developer website:

https://developer.ogc.org

You can learn more about past and upcoming developer events (including code sprints) in the developer events wiki page:

https://github.com/opengeospatial/developer-events/wiki

15. Conclusion

We hope this workshop provided a valuable overview of OGC APIs, with the goal of enabling low barrier, simple and flexible geospatial data publishing in support of FAIR data principles.

16. Contributing

Suggestions, improvements and fixes are always welcome. Please visit our GitHub page for more information on getting in touch.

Thank you for your interest in OGC APIs!