OGC API - Features

Clemens Portele
Panagiotis (Peter) Vretanos

The world's leading and comprehensive community of experts making location information:

#OGCAPI

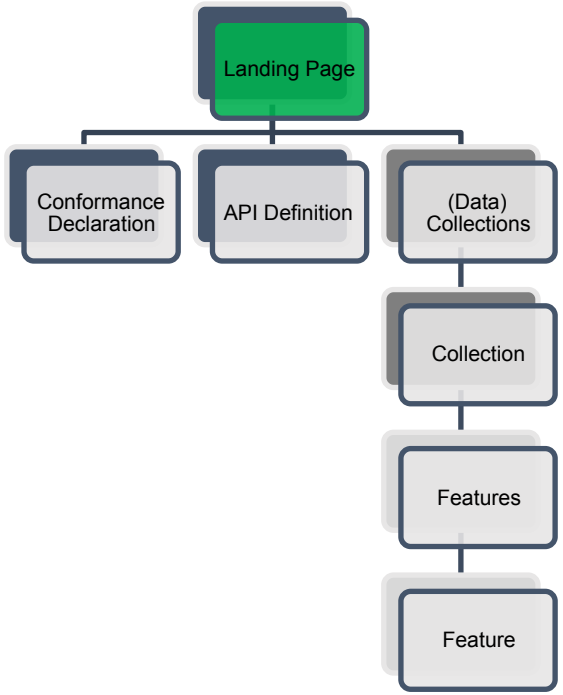
- **F**indable
- **A**ccessible
- **I**nteroperable
- **R**eusable

Current OGC API Features parts

- Part 1: Core
  - *published in October 2019, conformance test available, same as ISO 19168-1:2020*
  - The API capabilities that almost every one needs
- Part 2: Coordinate Reference Systems by Reference
  - *published in October 2020, conformance test available, same as ISO/DIS 19168-2*
  - Support beyond WGS84 for well-known CRS
- Part 3: Filtering
  - *draft, close to the final release candidate*
  - Filter expressions on features from a single feature collection
- Part 4: Create, Replace, Update and Delete
  - *draft, needs more implementations and testing*
  - Mutations, one feature per request

https://github.com/opengeospatial/ogcapi-features/blob/master/CHARTER.adoc#scope-of-work

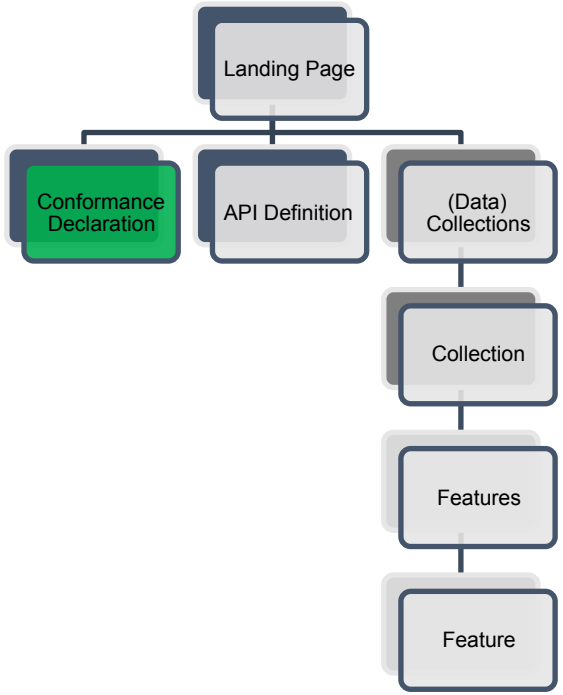
Resources: Landing Page



- Starting point to navigate the OGC API resources in this API
- Relative path: /

https://demo.ldproxy.net/zoomstack

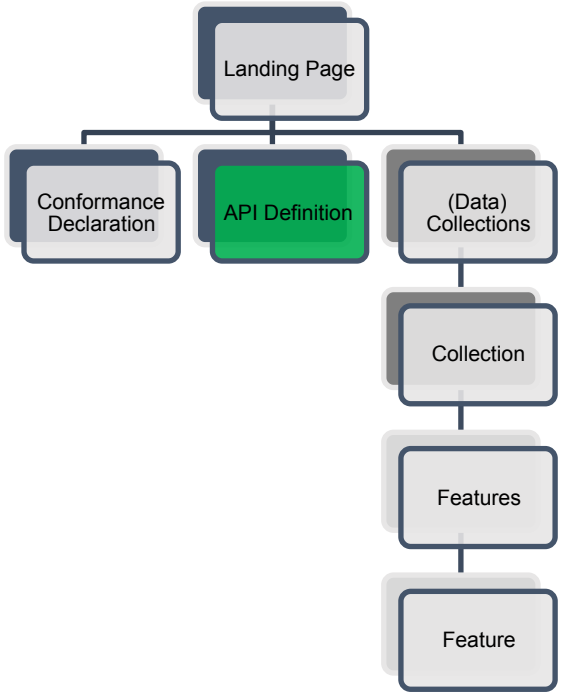
Resources: Conformance Declaration



- Lists the conformance classes from standards or community specifications, identified by a URI, that the API conforms to
- Relative path: `/conformance`

https://demo.ldproxy.net/zoomstack/conformance

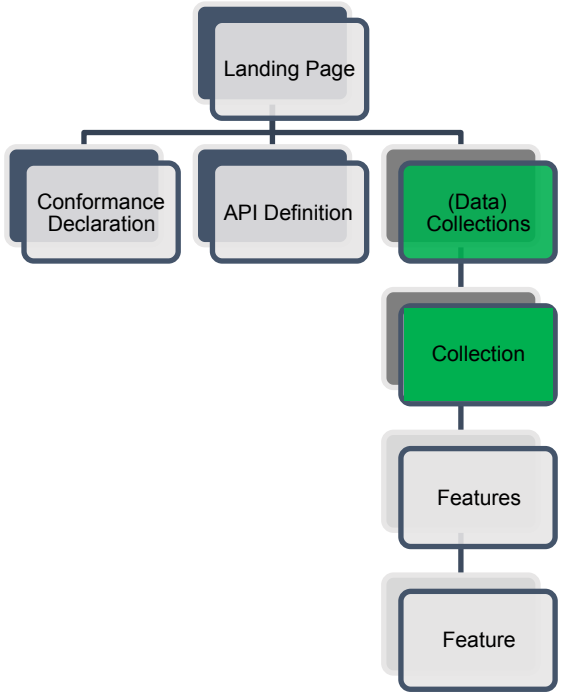
Resources: API Definition



https://demo.ldproxy.net/zoomstack/api?f=html
https://demo.ldproxy.net/zoomstack/api?f=json
https://demo.ldproxy.net/zoomstack/api?f=yaml

- Description/documentation of the API including the Landing Page and its sub-resources
- No fixed path
  - often `/api`
  - may also be external, e.g. on SwaggerHub
- OpenAPI document or an alternative service description

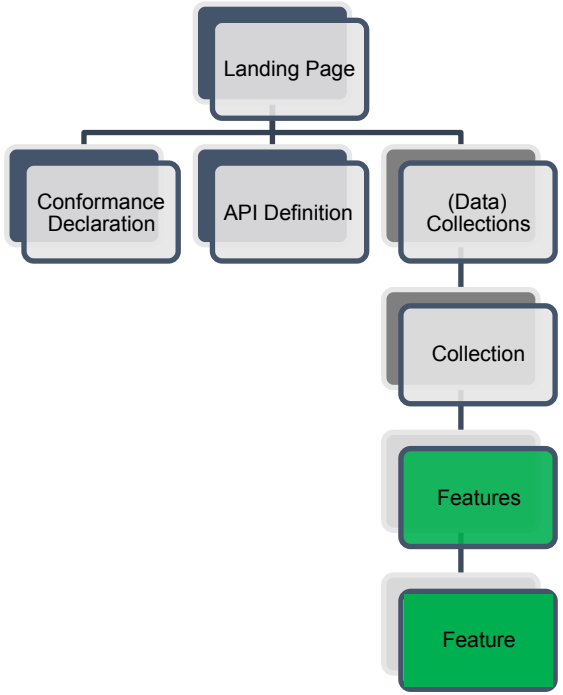
Resources: Collections



- The data in a dataset is organized into one or more collections
  - Information to generate meaningful requests to features (spatial/temporal extent, CRS, etc.)
- Relative path: `/collections/{collectionId}`

https://demo.ldproxy.net/zoomstack/collections
https://demo.ldproxy.net/zoomstack/collections/airports

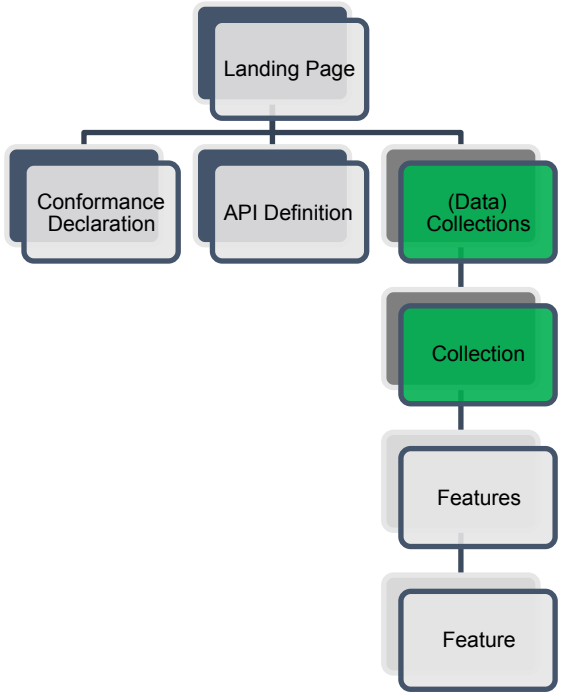
Resources: Features



- Paged access to features in a collection and a URI for every feature
- Relative path: `/collections/{collectionId}/items/{featureId}`
- Parameters: `bbox`, `datetime`, `limit`, collection-specific parameters
  - All filtering predicates are combined with an implicit AND

https://demo.ldproxy.net/zoomstack/collections/airports/items
https://demo.ldproxy.net/zoomstack/collections/airports/items/1
https://demo.ldproxy.net/daraa/collections/CulturePnt/items?datetime=../2012-12-31T23:59:59Z&bbox=36.09,32.6,36.1,32.61&F_CODE=AL030

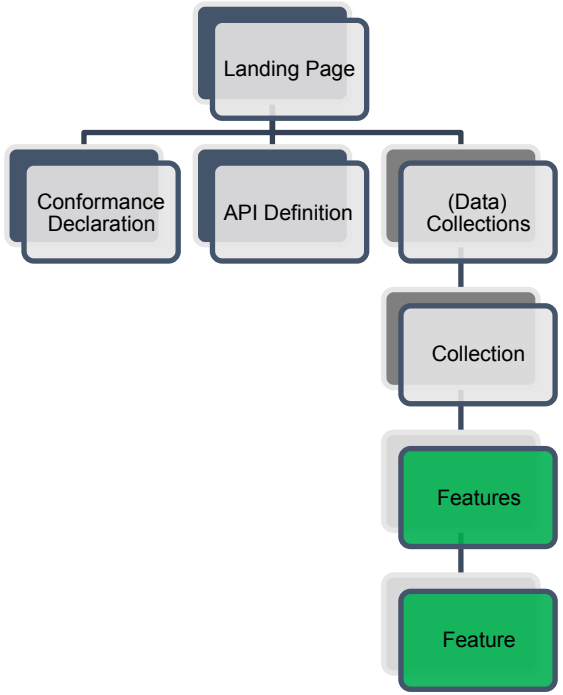
CRS (Part 2)



Adds

- the list of CRSs in which geometries can be requested,
- the CRS in which the data is stored (no transformation) and
- the epoch (in case of dynamic datums)

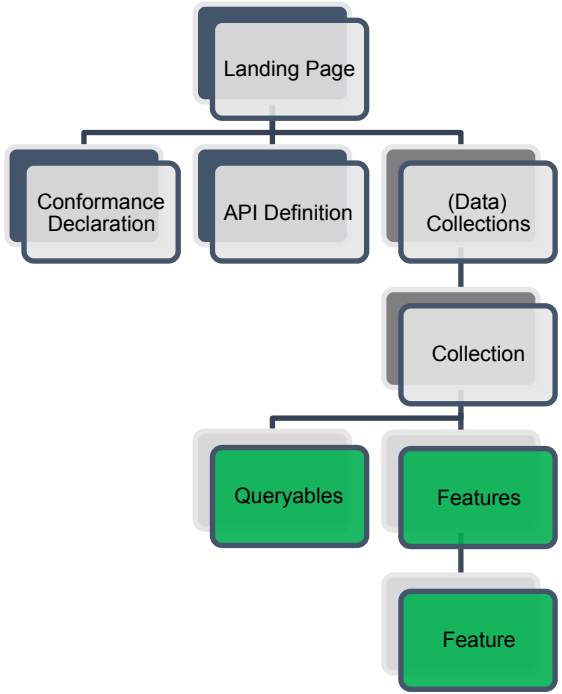
CRS (Part 2)



- Parameters:
  - `crs`: URI of the CRS to use in the response
  - `bbox-crs`: URI of the CRS in which the coordinates are in the bbox parameter; axis order follows the axis order in the CRS
- HTTP headers:
  - `Content-Crs`: Identifies the CRS used in the response

https://demo.ldproxy.net/zoomstack/collections/airports/items?crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FEPSG%2F0%2F27700&f=json
https://demo.ldproxy.net/zoomstack/collections/airports/items/1?crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FEPSG%2F0%2F27700&f=json
https://demo.ldproxy.net/zoomstack/collections/airports/items?bbox=50,-1,52,1&bbox-crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FEPSG%2F0%2F4326&crs=http%3A%2F%2Fwww.opengis.net%2Fdef%2Fcrs%2FEPSG%2F0%2F27700&f=json

Common Query Language (CQL2) Filtering (Part 3)

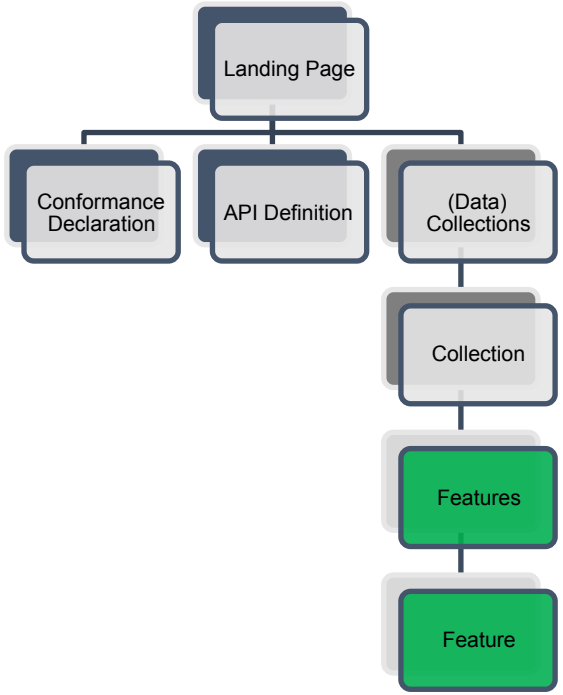


- Composed of 2 specs; generic part (CQL2) and feature-specific part (Part 3)
- Part 3 defines:
  - Queryables:
    - List of properties that can be used in filter expressions
  - Parameters:
    - `filter`: the filter expression
    - `filter-crs`: the CRS of geometries in the filter expression, default is WGS 84 longitude, latitude
    - `filter-lang`: the language used in the filter expression, default is `cql2-text`
- Common Query Language (CQL2) defines:
  - Basic CQL2 (literals, logical op's, comparison op's, property-value comparisons)
  - Advanced Comparison Operators (LIKE, BETWEEN, IN)
  - Case Insensitive Comparisons (ICASE,IACCENT)
  - Basic Spatial Operators (S_INTERSECTS)
  - Spatial Operators (all the others)
  - Temporal Operators (T_AFTER, T_OVERLAPS, etc.)
  - Array Operators (A_CONTAINEDBY, A_CONTAINS, etc.)
  - Property-Property Comparisons
  - Custom Functions
  - Arithmetic Expressions
  - 2 encodings defined: TEXT and JSON

Select all motorways changed after 2011:

```
F_CODE='AP030' AND
T_AFTER(ZI001_SDV, DATE('2011-12-31')) AND
RTY IN (1,2)
```

Create, Replace, Update, Delete (Part 4)



- Like Part 3 there is a generic components and a feature-specific part.
- The generic part defines:
  - **create/replace/delete** for inserting, replacing and deleting resources
  - **update** for modifying parts of resources
  - **optimistic locking** for handling concurency
- The feature specific part maps methods to features resources:

	POST	PUT	PATCH	DELETE
Features:	create	n/a	n/a	n/a
Feature:	n/a	replace	update	delete

Next OGC API Features work items

- Schemas
  - Schema resources for the features in a feature collection; will consider multiple types of schemas
- Queries/Search
  - Query resource supporting GET and POST; the Query resource at path “/search” will support multi-collection queries, the Query resource at path “/collections/{collectionId}/search” will support queries on a single collection; support for stored/ persistent queries will be considered
- Property Selection
  - One or more query parameters to reduce the properties that are returned for a feature (resources Features and Features)
- Geometry simplification
  - A query parameter to indicate a scale or zoom level at which the features will be displayed (resources Features and Features)
- OpenAPI 3.1
  - Add support for OpenAPI 3.1 including JSON Schema 2020-12

https://github.com/opengeospatial/ogcapi-features/blob/master/CHARTER.adoc#scope-of-work
https://github.com/opengeospatial/ogcapi-features/tree/master/proposals

More information

- OGC resources
  - https://ogcapi.ogc.org/features/
  - https://github.com/opengeospatial/ogcapi-features
- Implementations
  - https://www.ogc.org/resource/products/compliant?specid=1022
  - https://github.com/opengeospatial/ogcapi-features/tree/master/implementations
- Example APIs in production
  - https://api.weather.gc.ca/
  - https://ogc-api.nrw.de/

Related topic: JSON-FG

- Developers today prefer JSON over XML
- GeoJSON popular and widely supported
- OGC API Features implementations typically support GeoJSON
- But intentional limitations exist in GeoJSON that are an issue for some use cases:
  - Restricted to WGS 84 as Coordinate Reference System
  - Ellipsoidal metrics not supported
  - Points, line strings and polygons – no support for solids
  - Supports spatial, but not temporal geometries

JSON-FG – Approach

- Develop an OGC Features and Geometries JSON standard addressing the identified limitations
  - Avoid edge cases, focus on capabilities that are useful for many spatial experts
  - Additional capabilities could be added in the future, if there is broad support for the initial OGC Features and Geometries JSON in implementations
- Specify as a superset of GeoJSON
  - i.e., valid GeoJSON is also valid OGC Features and Geometries JSON
  - adding additional top-level members and links in the JSON objects (feature and feature collection)
- No dependency on JSON-LD
  - but for those that want to use JSON-LD, avoid conflicts
- GitHub repository:
  - https://github.com/opengeospatial/ogc-feat-geo-json

JSON-FG – Current extensions

- **"featureType"**: Features are often categorized by type and GIS clients often depend on knowledge about the feature type
- **"when"**: Temporal extent (instant or interval), matches the datetime parameter in OGC API Features
- **"where"**: Spatial geometries that are not in WGS 84
- **"coordRefSys"**: Declaration of the coordinate reference system used in "where" geometries
- **"links"**: Links to JSON schemas, other features, etc.

OGC Features API SWG Sprint Goals

Discussion and implementation feedback on topics that we are currently working on, in particular:

- JSON-FG: Develop proposal for a new conformance class (e.g., how to select fallback geometry or not), feedback on the proposed JSON-FG extensions
- Schemas: Work on guidance for client-friendly JSON schemas
- Geometry simplification: Use the current proposal and issues as a starting point
- CQL2: Additional implementations and feedback

Thank You!

Community

500+ International Members
110+ Member Meetings
60+ Alliance and Liaison partners
50+ Standards Working Groups
45+ Domain Working Groups
25+ Years of Not for Profit Work
10+ Regional and Country Forums

Innovation

120+ Innovation Initiatives
380+ Technical reports
Technology Forecasting to drive innovation

Standards

65+ Adopted Standards
300+ products with 1000+ certified implementations
1,800,000+ Operational Data Sets Using OGC Standards

Contact info@ogc.org to schedule a meeting for an in-depth discussion with OGC staff and join our community today!

Key characteristics, WFS comparison

OGC API - Features	OGC Web Feature Service 2.0 / ISO 19142
Modular, part 1 are the capabilities that almost everyone sharing a geospatial feature dataset will need	One standard for everything
Consistent with Web architecture	RPC with HTTP as a tunnel, HTTPS not officially supported
Reuse web standards where possible (mainly IETF, OAI)	OGC-specific capabilities, exception codes, etc.
Transactional capabilities added by part 4	Transactions: an optional conformance class
WGS84 only in part 1 (longitude, latitude, optional elevation), additional CRS's supported in part 2	Any CRS
Very simple filtering in part 3, more advanced capabilities added by part 3 and the future Search/Query extension	Support for Filter Encoding
No URI has a mandatory query parameter (meaningful defaults)	Knowledge about query parameters essential to create a valid request
Schemaless, schema support added in part 3, 4 and the future Schema extension	GML application schemas are mandatory
Features can be organized into collections as needed for the intended use of the API	Features are organized by feature type
No encoding is mandatory, but recommended encodings for all resources that can change with time (currently JSON and HTML)	XML mandatory
One landing page per dataset	Dataset is not a concept discussed in WFS
One API can provide other resources from the dataset, e.g. tiles, maps	Separate WMTS/WMS endpoints needed, with unspecified relationship