API Design Rules Module: Geospatial 1.0.4

Logius Guide
Definitive version June 04, 2025

This version:
https://gitdocumentatie.logius.nl/publicatie/api/mod-geo/1.0.4/

Latest published version:
https://gitdocumentatie.logius.nl/publicatie/api/mod-geo/

Latest editor's draft:
https://logius-standaarden.github.io/API-mod-geospatial/

Previous version:
https://gitdocumentatie.logius.nl/publicatie/api/mod-geo/1.0.3/

Editor:
Linda van den Brink (Geonovum)

Authors:
Pieter Bresters (Geonovum)
Linda van den Brink (Geonovum)
Paul van Genuchten (ISRIC)
George Mathijssen (Sweco Nederland B.V.)
Mark Strijker (Het Kadaster)

Participate:
GitHub Logius-standaarden/API-mod-geospatial
All issues
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is the definitive version of this document. Edits resulting from consultations have been applied.

Table of Contents

§Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

Abstract

This document is part of the *Nederlandse API Strategie*.

The Nederlandse API Strategie consists of a set of distinct documents.

Status	Description & Link
Informative	Inleiding NL API Strategie
Informative	Architectuur NL API Strategie
Informative	Gebruikerswensen NL API Strategie
Normative	NLgov REST API Design Rules (ADR v2.2)
Normative	Open API Specification (OAS 3.0)
Normative	NLgov Assurance profile for OAuth 2.0
Normative	Digikoppeling REST API koppelvlak specificatie
Normative module	GEO module v1.0

Before reading this document it is advised to gain knowledge of the informative documents, in particular the Architecture.

This document describes the *Geospatial module*, containing rules for geospatial content and functions in APIs.

§1. Introduction

This document provides rules for publishing geospatial data using Web APIs. Spatial data is

> data that describes anything with spatial extent (i.e. size, shape or position). Spatial data is also known as location information. [*sdw-bp*]

Geospatial data is more specific in that it is explicitly located relative to the Earth.

Geospatial data is 'special' data in the sense that it typically indicates the location of things using geometry. This geometry allows geospatial functions such as 'find only the things located within this area' but also requires specific ways of handling. There are international regulations and standards for geospatial data that need to be taken into account in certain cases.

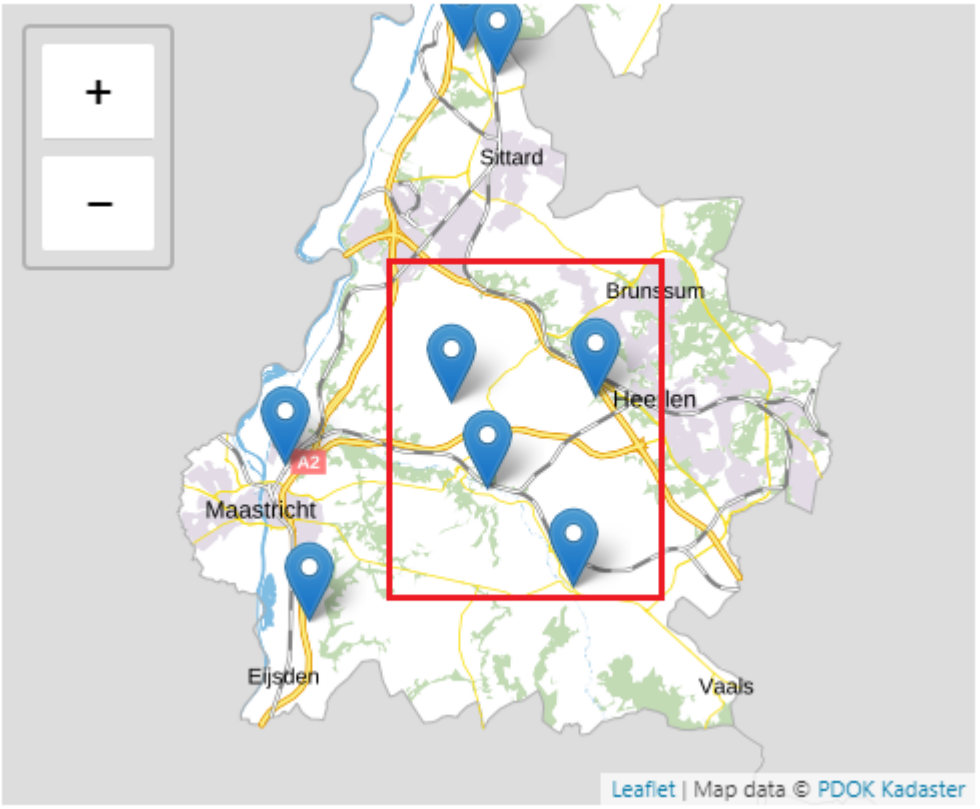

Figure 1 The red bounding box acts as a filter to find only the castles located within this area

The Geospatial Module provides rules for the structuring of geospatial payloads and for functions in APIs to handle geospatial data.

§1.1 Summary

Design rules are technical rules, which should be tested automatically, and functional rules, which should be considered when designing and building the API.

List of technical rules

- /geo/bbox-query-parameter: Supply a simple spatial filter as a bounding box parameter
- /geo/geometric-context: Place results of a global spatial query in the relevant geometric context
- /geo/geojson-request: Support GeoJSON in geospatial API requests
- /geo/embed-geojson-geometry-request: Embed GeoJSON Geometry object as part of the JSON resource in API requests
- /geo/geojson-response: Support GeoJSON in geospatial API responsess

- /geo/embed-geojson-geometry-response: Embed GeoJSON Geometry object as part of the JSON resource in API responses
- /geo/storage-crs: Make known in which CRS the geospatial data is stored by specifying the property storageCrs in the collection object
- /geo/default-crs: Use CRS84 as the default coordinate reference system (CRS) in line with OGC API Features Requirement 10
- /geo/ensemble-member-crs: The ensemble member should be one of the CRSs supported by the API
- /geo/bbox-crs-query-parameter: Support passing the coordinate reference system (CRS) of the bounding box in the request as a query parameter
- /geo/filter-crs-query-parameter: Support passing the coordinate reference system (CRS) of the geospatial filter in the request as a query parameter
- /geo/content-crs-request-header: When HTTP POST, PUT and/or PATCH requests are supported, pass the coordinate reference system (CRS) of geometry in the request body as a header
- /geo/crs-query-parameter: Support passing the desired coordinate reference system (CRS) of the geometry in the response as a query parameter
- /geo/content-crs-response-header: Assert the coordinate reference system (CRS) used in the response using a header

List of functional rules

- /geo/crs-list: Provide a list of all CRSs that are supported by the API
- /geo/preferred-crs: Use ETRS89 and/or RD when required

§2. Request and response

Providing requested resources is the essence of any API. This also applies to REST APIs that handle geospatial data. There are, however, some specific aspects when dealing with geospatial data in REST APIs. The most important aspects are described in this chapter:

- how to encode geometries in APIs
- how to supply a spatial filter in the call (request)
- how to return results of a spatial search

When requesting information, for example about cadastral parcels, users do not necessarily require the geometry, even if they used a spatial filter. A name or parcel ID may be sufficient.

> NOTE
> The Geospatial Module is focused on JSON-based encoding of data. However, consider also supporting `text/html`, as recommended in OGC API Features [*ogcapi-features-1*]. Sharing data on the Web should include publication in HTML, as this allows discovery of the data through common search engines as well as viewing the data directly in a browser.

§2.1 GeoJSON

[*RFC7946*] describes the GeoJSON format, including a convention for describing 2D geometric objects in CRS84 (OGC:CRS84). In the Geospatial module of the API strategy we adopt the GeoJSON conventions for describing geometry objects. The convention is extended to allow alternative projections. The GeoJSON conventions and extensions described in this module apply to both geometry passed in input parameters and responses.

> NOTE
> GeoJSON does not cover all use cases. For example, it is not possible to store circular arc geometries or solids in GeoJSON. In such cases, there are several valid options:
>
> - Use alternative standardized formats for geospatial data, such as WKT or its binary equivalent WKB; GML [iso-19136-2007]; or in future OGC JSON-FG (currently a draft standard).
> - When supporting GML, do this according to OGC API Features Requirements class 8.4 for GML Simple Features level 0, or Requirements class 8.4 for GML Simple Features level 2.
> - Use a workaround, e.g. convert circular lines / arcs to regular linestrings.

Example of embedding WKT in a JSON object using the following definition for a JSON object:

EXAMPLE 1

```
building:
  type: object
  required:
    - geometry
  properties:
    geometry:
      type: string
      format: wkt
```

Sample response:

EXAMPLE 2

```
{
  "building": {
    "geometry": "POLYGON Z((194174.445 465873.676 0, 194174.452 465872
  }
}
```

Example of embedding WKB in a JSON object using the following definition for a JSON object:

EXAMPLE 3

```
building:
  type: object
  required:
    - geometry
  properties:
    geometry:
      type: string
      format: wkb
```

Sample response:

EXAMPLE 4

```
{
  "building": {
    "geometry": "01030000A0F71C0000010000001200000F6285C8FF3B30741105
  }
}
```

§2.2 Call (requests)

A simple spatial filter can be supplied as a bounding box. This is a common way of filtering spatial data and can be supplied as a parameter. We adopt the OGC API Features [*ogcapi-features-1*] bounding box parameter:

Technical

/geo/bbox-query-parameter: Supply a simple spatial filter as a bounding box parameter

Statement

Support the OGC API Features part 1 `bbox` query parameter in conformance to the standard.

EXAMPLE 5

```
GET /api/v1/buildings?bbox=5.4,52.1,5.5,53.2
```

NOTE

Note that if a resource contains multiple geometries, it is up to the provider to decide if geometries of type single geometry or type multiple geometry are returned and that the provider shall clearly document this behavior.

Rationale

The default spatial operator `intersects` is used to determine which resources are returned.

Due to possible performance issue, especially when a combination of filters is used, a provider may decide to limit the size of the bounding box or the number of results. It is also up to the provider to decide if an error is returned in such cases. The provider shall clearly document this behavior.

The provider shall be able to provide resources that do not have a geometry property and are related to resources that match the bounding box filter.

An error shall be given if the provided coordinates are outside the specified coordinate reference system.

How to test

1. Issue an HTTP GET request to the API, including the `bbox` query parameter and using CRS Negotiation.
2. Validate that a response with status code 200 is returned.
3. Verify that only features that have a spatial geometry that intersects the bounding box are returned as part of the result set.

NOTE

Spatial operations like `intersects` and `within` in combination with a filter geometry (e.g. `bbox`) or resource geometry containing long lines, may result in erroneous responses, since a straight line between two coordinates in a CRS is, depending on the CRS, not a straight line in reality. See the explanation in the Handreiking CRS (Dutch).

NOTE

Spatial filtering is an extensive topic. There are use cases for geospatial operators like `intersects` or `within`. Geospatial filters can be large and complex, which sometimes causes problems since `GET` may not have a payload (although supported by some clients).

More complex spatial filtering is not addressed in this module. A new API Design Rules module on filtering will address spatial as well as non-spatial filtering. [*ogcapi-features-3*] will provide input for this.

However, until the filtering module is written, the geospatial module retains rule /geo/geometric-context about dealing with results of a global spatial query. This rule may be moved to the filtering module at a later stage.

Technical

/geo/geometric-context: Place results of a global spatial query in the relevant geometric context

Statement

In case of a global query `/api/v1/_search`, results should be placed in the relevant geometric context, because results from different collections, i.e. different sets of resources of the same type, are retrieved. Express the name of the collection

to which the results belong in the singular form using the property `type`. For example:

EXAMPLE 6

```
// POST /api/v1/_search:
{
  "currentPage": 1,
  "nextPage": 2,
  "pageSize": 10,
  "_embedded": {
    "items": [
      {
        "type": "enkelbestemming",
        "_links": {
          "self": {
            "href": "https://api.example.org/v1/enkelbeste
          }
        }
      },
      {
        "type": "dubbelbestemming",
        "_links": {
          "self": {
            "href": "https://api.example.org/v1/dubbelbest
          }
        }
      }
    ]
  }
}
```

How to test

1. Issue an HTTP GET request to the API.
2. Validate that the returned document contains the expected `type` property for each member.

In case a REST API shall comply to the OGC API Features specification for creating, updating and deleting a resource, the following applies.

Technical

/geo/geojson-request: Support GeoJSON in geospatial API requests

Statement

For representing geometric information in an API, use the convention for describing geometry as defined in the GeoJSON format [*RFC7946*]. Support GeoJSON as described in OGC API Features part 4, but note that this standard is still in development.

EXAMPLE 7

POST feature

```
// POST /collections/gebouwen/items   HTTP/1.1
// Content-Type: application/geo+json
{
  "type": "Feature",
  "geometry":  {
    "type": "Point",
    "coordinates": [5.2795,52.1933]
  },
  "properties": {
    "naam": "Paleis Soestdijk",
    // ...
  }
}
```

EXAMPLE 8

POST feature collection

```
// POST /collections   HTTP/1.1
// Content-Type: application/geo+json
{
  "type": "FeatureCollection",
  "features": [
  {
    "type": "Feature",
    "geometry":  {
      "type": "Point",
      "coordinates": [5.2795,52.1933]
    },
    "properties": {
      "naam": "Paleis Soestdijk",
      // ...
    }
  }]
}
```

How to test

1. Create a new resource that includes feature content (i.e. coordinates) using the HTTP POST method with request media type `application/geo+json` in the `Content-Type` header.
2. Validate that a response with status code `201` (Created) is returned.
3. Validate that the response includes the `Location` header with the URI of the newly added resource.

In case a REST API does not have to comply to the OGC API Features specification, e.g. for usage in administrative applications, the REST API shall use the JSON data format. If a resource contains geometry, that geometry shall be embedded as a GeoJSON `Geometry` object within the resource. The media type `application/json` must be supported. This may also apply to other media types `application/*+json`, however this depends on the media type specification. If the media type specification prescribes that resource information must be embedded in a JSON structure defined in the media specification, then the media type should not be supported while it is impossible to comply to that specification with the method described below. The media type `application/geo+json` should not be supported while the resource does not comply to the

GeoJSON specification, i.e. the request resource does not embed a feature or feature collection. A template for the definition of the schemas for the GeoJSON `Geometry` object in the requests in OpenAPI definitions is available: geometryGeoJSON.yaml. In case a collection of resources is embedded in the request resource, the name of the array containing the resources should be the plural of the resource name.

Technical

/geo/embed-geojson-geometry-request: Embed GeoJSON Geometry object as part of the JSON resource in API requests

Statement

When a JSON (`application/json`) request contains a geometry, represent it in the same way as the `Geometry` object of GeoJSON.

EXAMPLE 9

POST resource containing geometry

```
// POST /collections/gebouwen/items   HTTP/1.1
// Content-Type: application/json
{
  "naam": "Paleis Soestdijk",
  "geometrie": {
    "type": "Point",
    "coordinates": [5.2795,52.1933]
  }
}
```

EXAMPLE 10

POST resource containing geometry collection

```
// POST /collections/gebouwen/items   HTTP/1.1
// Content-Type: application/json
{
  "naam": "Paleis Soestdijk",
  "geometrie": {
    "type": "GeometryCollection",
    "geometries": [
      {
        "type": "Point",
        "coordinates": [5.2795,52.1933]
      }
    ]
  }
}
```

How to test

1. Create a new resource that includes geometry of GeoJSON `Geometry` object type using the HTTP POST method with request media type `application/json` in the `Content-Type` header.
2. Validate that a response with status code `201` (Created) is returned.
3. Validate that the response includes the `Location` header with the URI of the newly added resource.

§2.3 Result (response)

In case a REST API shall comply to the OGC API Features specification, e.g. for usage in GIS applications, the following applies.

/geo/geojson-response: Support GeoJSON in geospatial API responsess

Technical

Statement

For representing 2D geometric information in an API response, use the convention for describing geometry as defined in the GeoJSON format [*RFC7946*]. Support GeoJSON as described in OGC API Features Requirements class 8.3 [*ogcapi-features-1*].

EXAMPLE 11

Feature

```
// GET /collections/gebouwen/items/0308100000022041   HTTP
// Content-type: application/geo+json
{
  "type": "Feature",
  "id": "0308100000022041",
  "geometry":  {
    "type": "Point",
    "coordinates": [5.2795,52.1933]
  },
  "properties": {
    "naam": "Paleis Soestdijk",
    // ...
  },
  "links": [
    {
      "self": "/collections/gebouwen/items/030810000002204
    }
  ]
}
```

EXAMPLE 12

Feature collection

```
// GET /collections/gebouwen   HTTP 1.1
// Content-type: application/geo+json
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": "0308100000022041",
      "geometry":  {
        "type": "Point",
        "coordinates": [5.2795,52.1933]
      },
      "properties": {
        "naam": "Paleis Soestdijk",
        // ...
      },
      "links": [
        {
          "self": "/collections/gebouwen/0308100000022041"
        }
      ]
    },
    {
    }
  ],
  "timeStamp" : "2023-02-22T10:32:23Z",
  "numberMatched" : "0308100000022041",
  "numberReturned" : "1",
  "links": [
    {
      "self": "/collections/gebouwen"
    },
    {
      "next": ""
    }
  ]
}
```

NOTE

The resources' properties (e.g. `naam`) are passed in the properties object. Depending on the implemented filter capabilities the properties object may contain all or a selection of the resources' properties.

The OGC API Features specification provides the possibility to add an array of links to a feature and feature collection, which may contain a self link and in case of a feature collection may contain navigation links

How to test

Test case 1:

1. Request a single resource that includes feature content (i.e. coordinates) with response media type `application/geo+json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/geo+json`
4. Validate that the returned document is a GeoJSON Feature document.

Test case 2:

1. Request a collection of resources that includes feature content (i.e. coordinates) with response media type `application/geo+json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/geo+json`
4. Validate that the returned document is a GeoJSON FeatureCollection document.

Test case 3:

1. Request a single resource that does not include feature content (i.e. coordinates) with response media type `application/geo+json` or `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/json`
4. Validate that the returned document is a JSON document.

Test case 4:

1. Request a collection of resources that do not include feature content (i.e. coordinates) with response media type `application/geo+json` or `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.

3. Validate that `Content-Type` header contains `application/json`
4. Validate that the returned document is a JSON document.

In case a REST API does not have to comply to the OGC API Features specification, e.g. for usage in administrative applications, the REST API shall use the JSON data format. If resources contain geometry, the geometry shall be returned as a GeoJSON `Geometry` object embedded in the resource. The media type `application/json` must be supported. This may also apply to other media types `application/*+json`, however this depends on the media type specification. If the media type specification prescribes that resource information must be embedded in a JSON structure defined in the media type specification, then the media type should not be supported while it is impossible to comply to that specification with the method described below. The media type `application/geo+json` should not be supported while the resource does not comply to the GeoJSON specification, i.e. the response does not return a feature or feature collection. A template for the definition of the schemas for the GeoJSON `Geometry` object in the responses in OpenAPI definitions is available: geometryGeoJSON.yaml. In case a collection of resources is returned, the name of the array containing the resources should be the plural of the resource name.

Technical

/geo/embed-geojson-geometry-response: Embed GeoJSON Geometry object as part of the JSON resource in API responses

Statement

When a JSON (`application/json`) response contains a geometry, represent it in the same way as the `Geometry` object of GeoJSON.

EXAMPLE 13

Resource containing geometry

```
// GET /gebouwen/0308100000022041   HTTP 1.1
// Content-type: application/hal+json
{
  "identificatie": "0308100000022041",
  "naam": "Paleis Soestdijk",
  "geometrie":  {
    "type": "Point",
    "coordinates": [5.2795,52.1933]
  },
  "_links": {
    {
      "self": "/gebouwen/0308100000022041"
    }
  }
}
```

EXAMPLE 14

Resource containing geometry collection

```
// GET /gebouwen/0308100000022041   HTTP 1.1
// Content-type: application/hal+json
{
  "identificatie": "0308100000022041",
  "naam": "Paleis Soestdijk",
  "geometrie": {
    "type": "GeometryCollection",
    "geometries": [
      {
        "type": "Point"
        "coordinates": [5.2795,52.1933]
      },
      {
        "type": "Polygon"
        "coordinates" : [/* ... */]
      }
    ]
  },
  "_links": {
    {
      "self": "/gebouwen/0308100000022041"
    }
  }
}
```

EXAMPLE 15

Collection of resources containing geometry

```
// GET /gebouwen   HTTP 1.1
// Content-type: application/hal+json
{
  "gebouwen": [
    {
      "identificatie": "0308100000022041",
      "naam": "Paleis Soestdijk",
      "geometrie":  {
        "type": "Point",
        "coordinates": [5.2795,52.1933]
      }
      "_links": {
        {
          "self": "/gebouwen/0308100000022041"
        }
      }
    }
  ],
  "_links": {
    {
      "self": "/gebouwen"
    },
    {
      "next": ""
    }
  }
}
```

NOTE

The resource and resource collection may be [*HAL*] resources and therefore may contain a `_links` object. The `_links` object should contain a self link and in case of a collection also navigation links (e.g. first, next prev, last). In such cases the `application/hal+json` media type may be used.

How to test

Test case 1:

1. Request a single resource that contains geometry of GeoJSON `Geometry` object type: `Point`, `MultiPoint`, `LineString`, `MultiLineString`, `Polygon` or `MultiPolygon` and with response media type `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/json`
4. Validate that the returned document is a JSON document.
5. Validate that the returned document contains a property that complies to one of the GeoJSON `Geometry` objects mentioned above and contains:
    - a property `type` containing the name of one of the GeoJSON `Geometry` object types mentioned above, and
    - a property `coordinates` containing an array with the coordinates. Depending on the type of geometry object, the content of the array differs.

Test case 2:

1. Request a collection of resources that contain geometry of GeoJSON `Geometry` object type: `Point`, `MultiPoint`, `LineString`, `MultiLineString`, `Polygon` or `MultiPolygon` and with response media type `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/json`
4. Validate that the returned document is a JSON document.
5. Validate that the returned document contains an array of resources and that each resource contains a property that complies to one of the GeoJSON `Geometry` objects mentioned above and contains:
    - a property `type` containing the name of one of the GeoJSON `Geometry` object types mentioned above, and
    - a property `coordinates` containing an array with the coordinates. Depending on the type of geometry object, the content of the array differs.

Test case 3:

1. Request a single resource that contains geometry of GeoJSON `Geometry` object type: `GeometryCollection` and with response media type `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/json`

4. Validate that the returned document is a JSON document.
5. Validate that the returned document contains a property that complies to the GeoJSON `Geometry` object mentioned above and contains:
    - a property `type` containing the name of the GeoJSON `Geometry` object type: `GeometryCollection`, and
    - a property `geometries` containing an array of GeoJSON `Geometry` objects.

Test case 4:

1. Request a collection of resources that contain geometry of GeoJSON `Geometry` object type: `GeometryCollection` and with response media type `application/json` in the `Accept` header.
2. Validate that a response with status code 200 is returned.
3. Validate that `Content-Type` header contains `application/json`
4. Validate that the returned document is a JSON document.
5. Validate that the returned document contains an array of resources and that each resource contains a property that complies to the GeoJSON `Geometry` object mentioned above and contains:
    - a property `type` containing the name of the GeoJSON `Geometry` object type: `GeometryCollection`, and
    - a property `geometries` containing an array of GeoJSON `Geometry` objects.

§3. Coordinate Reference System (CRS)

A Coordinate Reference System (CRS) or Spatial Reference System (SRS) is a framework to measure locations on the earth surface as coordinates. Geometries consist of coordinates. To be able to measure the geometry's coordinates on the earth surface a CRS is required in conjunction with the coordinates.

CRSs are uniquely identified by means of a Spatial Reference System Identifier (SRID). SRIDs may refer to different standards, for example EPSG Geodetic Parameter Dataset or Open Geospatial Consortium (OGC).

CRSs may be grouped into ensemble CRSs, e.g. ETRS89 (EPSG:4258). The CRSs that are part of an ensemble CRS are called ensemble member CRSs or member CRSs that realize a ensemble

CRS, e.g ETRF2000 (EPSG:9067) is a member of and realizes the ETRS89 (EPSG:4258) ensemble. When exchanging geometry an ensemble member CRS shall be used instead of an ensemble CRS when known and if accurate data is required. When transforming geometry from one CRS to another, use an ensemble member CRS (instead of an ensemble CRS) as input and output of coordinate transformation, when known and if accurate data is required.

For a detailed description of CRSs see [*hr-crs*].

> NOTE
> In the geospatial module the abbreviation RD is used. RD refers to the "Stelsel van de Rijksdriehoeksmeting", this is the equivalent of EPSG code 28992 and EPSG name Amersfoort / RD New.

§3.1 CRS discovery

A client shall be able to determine a list of CRSs supported by an API.

Functional

/geo/crs-list: Provide a list of all CRSs that are supported by the API

Statement

If a REST API shall comply to the OGC API Features specification then the API must provide an endpoint to determine a list of supported CRSs.

> EXAMPLE 16
> `GET /api/v1/collections`

If a REST API does not have to comply to the OGC API Features specification, e.g. when the API is used for administrative purposes, then the API shall also provide an endpoint to determine the supported CRSs.

> EXAMPLE 17
> `GET /api/v1/crss`

Rationale

According to OGC API Features - part 1 - 7.13. Feature collections an OGC API Features API shall provide a GET operation on the `/collections` endpoint which returns a collections object.

OGC API Features - part 2 - Coordinate Reference Systems by Reference [*ogcapi-features-2*] describes how to support different CRSs in your geospatial API. According to OGC API Features - part 2 - 6.2 Discovery and in particular Global list of CRS identifiers, a collections object provided by the API's `/collections` endpoint may contain a global list of supported CRSs by means of the `crs` property. This global CRS list applies to all feature collections delivered by the API, unless otherwise stated at a feature collection.

Each feature collection mentioned within the `collections` list may also contain a `crs` property if the set of supported CRSs differs from the global CRS list. If a feature collection supports exactly the same CRSs as mentioned in the global CRS list, then the `crs` property may be omitted.

If a feature collection supports additional CRSs compared to the global CRS list in the collections object, then a reference to the global CRS list `#/crs` may be added in the feature collection object and the URIs of the additional CRSs are added to the CRS list in the `crs` property of the feature collection.

If a feature collection supports a different set of CRSs than the set defined in the global CRS list, then a reference to the global CRS list is omitted and only the URIs of the supported CRSs are added to the CRS list in the `crs` property of the feature collection.

For clients, it may be helpful to know the CRS identifier that may be used to retrieve features from that collection without the need to apply a CRS transformation. If all features in a feature collection are stored using a particular CRS, the property `storageCRS` shall be used to specify this CRS, in accordance with OGC API Features - part 2 - 6.2.2 Storage CRS. The value of this property shall be one of the CRSs supported by the API and advertised in the CRS list as stated in requirement 4 of OGC API Features - part 2 - 6.2.2 Storage CRS. If relevant, the epoch should also be specified, using the `storageCRSCoordinateEpoch` property. For an explanation of the use of epochs with CRS, see the CRS Guidelines [*hr-crs*].

How to test

If the REST API shall comply to the OGC API Features specification:

1. Issue an HTTP GET request to the `/collections` endpoint of the API.
2. Validate that the returned document contains a `collections` object with the `crs` property.

If the REST API does not have to comply to the OGC API Features specification:

1. Issue an HTTP GET request to the `/crss` endpoint of the API.

2. Validate that the returned document contains an object with a `crs` property.

In both cases, verify the following based on the response:

1. Validate that the `crs` property contains an array with CRS references in the form of URIs.
2. Validate that the CRS URIs return a GML document with an `epsg:CommonMetadata` element (`xmlns:epsg="urn:x-ogp:spec:schema-xsd:EPSG:1.0:dataset`).

Technical

/geo/storage-crs: Make known in which CRS the geospatial data is stored by specifying the property storageCrs in the collection object

Statement

The value of this property shall be one of the CRSs the API supports.

How to test

1. Issue an HTTP GET request to each collection in the `/collections` endpoint of the API.
2. Validate that each returned collection contains the `storageCRS` property.
3. Validate that the value of the `storageCRS` property is one of the URIs from the list of supported CRSs.

3.2 CRS negotiation

The default CRS for GeoJSON and for OGC API Features is CRS84 (OGC:CRS84), this CRS uses the WGS 84 datum with an ellipsoidal coordinate system in the order longitude-latitude. This refers to an ensemble of global CRSs that can be applied world-wide. For accurate applications the use of the CRS84 ensemble is not suitable. For more information about coordinate reference systems, read the Geonovum guidelines on CRS [*hr-crs*].

NOTE

When referring to a coordinate reference system using its code in the rest of this chapter, this is meant to refer to both the 2D and 3D variant of the system in question. E.g. when "RD" is mentioned, this should be taken to mean "RD or RDNAP"; when WGS 84 is mentioned, this should be taken to mean "CRS84 or CRS84h". Each variant has an identifier.

Technical

/geo/default-crs: Use CRS84 as the default coordinate reference system (CRS) in line with OGC API Features Requirement 10

Statement

The implication of this is, that if no CRS is explicitly included in the request, CRS84 is assumed. This rule also applies if the request uses POST.

Rationale

Since most client-side mapping libraries use WGS 84 longitude-latitude (CRS84), the W3C/OGC Spatial Data on the Web working group recommends to use this as the default coordinate reference system. The API strategy caters for this supporting not only ETRS89 and RD, but also CRS84.

The *default* CRS, i.e. the CRS which is assumed when not specified by either the API or the client, is CRS84, in line with GeoJSON and OGC API Features.

How to test

1. Issue an HTTP GET request to retrieve some spatial data from the API without specifying a coordinate reference system.
2. Validate that the response includes a `Content-Crs` header with the URI for CRS84 or CRS84h.
3. Validate the retrieved spatial data using the CRS84 reference system (for 2D geometries) or the CRS84h reference system (for 3D geometries).

Functional

/geo/preferred-crs: Use ETRS89 and/or RD when required

Statement

These are the preferred coordinate reference systems (CRS) for Dutch geospatial data. General usage of the European ETRS89 coordinate reference system (CRS) or RDNAP is preferred, but is not the default CRS. Hence, one of these CRSs has to be explicitly included in each request when one of these CRSs is desired in the response or used in a request. This is part of the Dutch Guideline for the use of CRSs [*hr-crs*].

How to test

1. Issue an HTTP GET request to retrieve some spatial data from the API, specifying ETRS89 and/or RD as coordinate reference system.
2. Validate that the response includes a `Content-Crs` header with the URI for the requested CRS.

3. Validate the retrieved spatial data using the coordinate reference system used in the request.

The guiding principles for CRS support:

- Source systems record coordinates as they enter the system;
- The default CRS, CRS84, is listed first in the list of supported CRSs in the API; if the consumer does not specify the CRS it is assumed it uses the default.
- Coordinate reference systems API strategy: request/response in RD; ETRS89; CRS84;
- Use the latest version of RDNAPTRANS™ to transform RD to ETRS89 (correction grid);
- Which CRSs are supported in an API depends on context (e.g. user requirements) - see Spatial Data on the Web Best Practice 7: Choose coordinate reference systems to suit your user's applications [*sdw-bp*];
- Exchange format (notation) for ETRS89 and CRS84 (longitude, latitude) in decimal degrees, for example: (`5.96237626, 52.25502345`). The longitude and latitude are decimal numbers. The number of decimals in the fractional part may vary depending on the required accuracy. For an accuracy of 1 mm, 8 decimals in the fractional part are sufficient. See Nauwkeurigheid van coördinaten in [*hr-crs*].
- Exchange format (notation) for RD (X, Y) in meters, for example: (`195427.520, 311611.840`). The X and Y coordinates are decimal numbers. The number of decimals in the fractional part may vary depending on the required accuracy. For an accuracy of 1 mm, 3 decimal places in the fractional part are sufficient. See Nauwkeurigheid van coördinaten in [*hr-crs*].
- WGS 84 Pseudo Mercator (EPSG:3857) is rather inaccurate, but suitable for simple visualization of inprecise spatial data on the web, e.g. when it suffices if the data is recognizable on a map. WGS 84 Pseudo Mercator shall not be used for precise data that is meant for accurate spatial analysis.
- Use the CRS Guidelines [*hr-crs*] for coordinate transformations.
- Use an ensemble member CRS (instead of an ensemble CRS) for exchanging geometry, when known.
- Use an ensemble member CRS (instead of an ensemble CRS) as output of coordinate transformation, when known.
- APIs shall support and advertise both ensemble CRSs and ensemble member CRSs if geometry is exchanged and the CRS for the geometry is an ensemble member CRS.
- Under certain conditions WGS 84 can be made equal to e.g. ETRS89, this is called a 'null transformation', see [*hr-crs*]. If a null transformation is used to realize WGS 84, then the CRS

(e.g. ETRS89) that is used to realize WGS 84 and the CRS for WGS 84 itself shall both be supported and advertised by an API.

Technical

/geo/ensemble-member-crs: The ensemble member should be one of the CRSs supported by the API

Statement

When the API provides data in an ensemble CRS like WGS 84 or ETRS89, while it is known to what ensemble member CRS the data actually refers, it should also be one of the CRSs supported by the API and advertised in the CRS list. For example when 2D data is transformed from RD with RDNAPTRANS not only ETRS89 (EPSG:4258) should be supported but also ETRF2000 (EPSG::9067).

How to test

1. Issue an HTTP GET request to the `/collections` endpoint.
2. Validate that the returned document contains a `collections` object with the `crs` property.
3. Validate that the `crs` property contains an array with CRS references in the form of URIs.
4. Validate that when the `crs` property contains a URL for a ensemble CRS like ETRS89 (EPSG:4258), it also contains a URL for a ensemble member CRS like ETRF2000 (EPSG:9067).

Technical

/geo/bbox-crs-query-parameter: Support passing the coordinate reference system (CRS) of the bounding box in the request as a query parameter

Statement

Support the OGC API Features part 2 `bbox-crs` parameter in conformance to the standard.

If a bounding box is sent to the server without these parameters, the default CRS, CRS84, is assumed as specified in /geo/default-crs.

If an invalid value, i.e. a CRS which is not in the list of supported CRSs, is given for one of these parameters, the server responds with an HTTP status code `400`.

How to test

1. Issue an HTTP GET request to the API, including the `bbox` parameter AND the `bbox-crs` parameter.
2. Validate that a document was returned with a status code 200.

3. Verify that the response includes a `Content-Crs` HTTP header with the URI of the requested CRS identifier.

Technical

/geo/filter-crs-query-parameter: Support passing the coordinate reference system (CRS) of the geospatial filter in the request as a query parameter

Statement

Support the OGC API Features part 3 `filter-crs` parameter in conformance to the standard.

If a geospatial filter is sent to the server without these parameters, the default CRS, CRS84, is assumed as specified in /geo/default-crs.

If an invalid value, i.e. a CRS which is not in the list of supported CRSs, is given for one of these parameters, the server responds with an HTTP status code `400`.

How to test

1. Issue an HTTP GET request to the API, including a geospatial filter AND the `filter-crs` parameter.
2. Validate that a document was returned with a status code 200.
3. Verify that the response includes a `Content-Crs` HTTP header with the URI of the requested CRS identifier.

In an API that supports the creation and/or updating of items, POST, PUT or PATCH requests with geospatial content in the body may be sent by a client to the server. In that case, it is necessary to indicate the CRS used, unless CRS84 (OGC:CRS84), the default CRS, is used.

Technical

/geo/content-crs-request-header: When HTTP POST, PUT and/or PATCH requests are supported, pass the coordinate reference system (CRS) of geometry in the request body as a header

Statement

Support the OGC API Features part 4 `Content-Crs` header in conformance to the standard.

Alternatively, if the feature representation supports expressing CRS information for each feature / geometry, the information can also be included in the feature representation. If no CRS is asserted, the default CRS, CRS84, is assumed, as stated in /geo/default-crs.

How to test

In a request (i.e. when creating or updating an item on the server):

1. Issue an HTTP POST request to the API with spatial data in the request body, including the `Content-Crs` header with the value of the CRS identifier for the spatial data in the body.
2. Verify that a document was returned with status code `201` in case a new item was created, or with status code `200`.

Repeat with a similar test voor PUT and/or PATCH if the server supports these.

Technical

/geo/crs-query-parameter: Support passing the desired coordinate reference system (CRS) of the geometry in the response as a query parameter

Statement

Support the OGC API Features part 2 `crs` parameter in conformance to the standard.

How to test

1. Issue an HTTP GET request to the API, including the `crs` parameter.
2. Verify that the response has the status code `200`, and includes a `Content-Crs` http header with the value of the requested CRS identifier.

Technical

/geo/content-crs-response-header: Assert the coordinate reference system (CRS) used in the response using a header

Statement

Support the OGC API Features part 2 `Content-Crs` header in conformance to the standard.

How to test

1. Issue an HTTP GET request to the API, requesting spatial data.
2. Verify that the response includes the `Content-Crs` header with the URI of the requested CRS identifier if explicitly requested, or with the value `http://www.opengis.net/def/crs/OGC/1.3/CRS84` if no CRS was explicitly requested.

The API should be able to handle the following scenarios based on the rules stated above:

Scenario	Explanation
No geometry in request, no geometry in response	No CRS negotiation necessary
No geometry in request, geometry in response	The client can request a specific CRS for the geometries in the response using the `crs` parameter. The server indicates the geometry CRS in the response using the `Content-Crs` header.
Geometry in request body, no geometry in response	The client indicates the CRS of the geometry in the request body using the `Content-Crs` header.
Geometry in request body, geometry in response	The client indicates the CRS of the geometry in the request body using the `Content-Crs` header, and can request a specific CRS for the geometries in the response using the `crs` parameter. The server indicates the geometry CRS in the response using the `Content-Crs` header.
Geometry filter in request, no geometry in response	The client indicates the CRS of the geometry filter in the request using the `bbox-crs` parameter if a bounding box is used to filter geospatially, or the `filter-crs` parameter if another way of geospatial filtering is used.
Geometry filter in request, geometry in response	The client indicates the CRS of the geometry filter in the request using `bbox-crs` or `filter-crs` as in the previous scenario, and requests a specific CRS for the geometries in the response using the `crs` parameter. The server indicates the geometry CRS in response using the `Content-Crs` header.

Below is a list of the most commonly used CRSs in the Netherlands:

Name	Code	Type	Dimension	Scope	URI
Amersfoort / RD New	28992	easting, northing (x, y)	2D	Dutch	
Amersfoort / RD New + NAP height	7415	easting, northing, height (x, y, h)	3D	Dutch	
ETRS89	4258	latitude, longitude (φ, λ)	2D	European	
ETRS89	4937	latitude, longitude, height (φ, λ, h)	3D	European	
ETRF2000	7931	latitude, longitude, height (φ, λ, h)	3D	European	

Name	Code	Type	Dimension	Scope	URI
ETRF2000	9067	latitude, longitude (φ, λ)	2D	European	
ITRF2014	7912	latitude, longitude, height (φ, λ, h)	3D	Global	
ITRF2014	9000	latitude, longitude (φ, λ)	2D	Global	
WGS 84 longitude-latitude	CRS84	longitude, latitude (λ, φ)	2D	Global	
WGS 84 longitude-latitude-height	CRS84h	longitude, latitude, height (λ, φ, h)	3D	Global	
WGS 84 / Pseudo-Mercator	3857	easting, northing (x, y)	2D	Global	

For a more extensive overview of CRSs see: https://docs.geostandaarden.nl/crs/crs/#bijlage-a-crs-overzicht-tabel. Note that the URI of each CRS contains a version number and that new versions may be released in future. Before using a URI verify if newer versions are available and use the latest version.

NOTE
New ensemble member CRSs may be released in future (e.g. ITRF2020 has been released as a realization for ITRF). These new realizations shall be used instead of older realizations in case of ITRF. In case of ETRF however, National Mapping Agencies have agreed on using ETRF2000 instead of newer realisations (e.g. ETRF2014).

NOTE
Officially, WGS 84 longitude-latitude (OGC:CRS84) is the only CRS allowed in GeoJSON. However, GeoJSON does state that using another CRS is allowed, if this is agreed between provider and consumer of the data. The API functionality described above, to negotiate the CRS between client and server, can be viewed as such an agreement. Many GIS clients can deal with GeoJSON in other CRS than CRS84 (OGC:CRS84).

In addition, the Geonovum CRS guidelines [*hr-crs*] describe how ETRS89 can be treated as equal to CRS84 (OGC:CRS84)under certain circumstances.

[*JSON-FG*] is a proposed standard extension of GeoJSON that adds CRS support.

§3.3 CRS transformation

If the requested CRS is not the same as the storage CRS, a coordinate transformation is needed. Performance is increased when the dataset is transformed in multiple CRSs and stored in advance, and not transformed at the moment the request has arrived. In case of a transformation between RD and ETRS89, it is required that this transformation uses the latest version of the procedure of RDNAPTRANS™.

§4. INSPIRE requirements

INSPIRE is a European directive that forces data providers of geospatial datasets that belong to one of the 34 INSPIRE themes to publish the metadata, a viewservice and a download service. These services can also be APIs.

For the OGC-API Features, an endorsed good practice has been described in a document that proposes a technical approach for implementing the requirements set out in the INSPIRE Implementing Rules for Network Services [IRs for NS] based on the newly adopted OGC API - Features standard.

The extra requirements stated in this document concern:

- links to predefined data set download
- multilinguality
- GeoJSON
- link to bulk download
- INSPIRE-CRS
- describing encoding
- metadata links

These requirements should be met when an API serves features for an INSPIRE dataset.

A. References

A.1 Normative references

[hr-crs]
Handreiking Gebruik coördinaatreferentiesystemen bij uitwisseling en visualisatie van geo-informatie. Lennard Huisman; Friso Penninga. Geonovum. Vastgesteld. URL: https://docs.geostandaarden.nl/crs/crs/

[ogcapi-features-1]
OGC API - Features - Part 1: Core. Open Geospatial Consortium. Approved. URL: http://docs.ogc.org/is/17-069r3/17-069r3.html

[ogcapi-features-2]
OGC API - Features - Part 2: Coordinate Reference Systems by Reference. Open Geospatial Consortium. Approved. URL: https://docs.ogc.org/is/18-058/18-058.html

[RFC7946]
The GeoJSON Format. H. Butler; M. Daly; A. Doyle; S. Gillies; S. Hagen; T. Schaub. IETF. August 2016. Proposed Standard. URL: https://www.rfc-editor.org/rfc/rfc7946

[sdw-bp]
Spatial Data on the Web Best Practices. Payam Barnaghi; Jeremy Tandy; Linda van den Brink; Timo Homburg. W3C. 19 September 2023. DNOTE. URL: https://www.w3.org/TR/sdw-bp/

A.2 Informative references

[HAL]
HAL - Hypertext Application Language. Mike Kelly. 2013-09-18. URL: http://stateless.co/hal_specification.html

[JSON-FG]
OGC Features and Geometries JSON - Part 1: Core. Open Geospatial Consortium. Editor's Draft. URL: https://docs.ogc.org/DRAFTS/21-045.html

[ogcapi-features-3]
OGC API - Features - Part 3: Filtering. Open Geospatial Consortium. Draft. URL: https://docs.ogc.org/is/19-079r2/19-079r2.html

↑