# RODEO EDR Profile

# The provision of Open Access to Public meteorological Data and Development of Shared federated Data Infrastructure for the Development of information Products and Services

Date: 2025-06-12

Version: 0.1.0

Document location: TBD

Document status: DRAFT

RODEO project partners[1]

# Table of Contents


1. Scope . . . 5
2. Conformance . . . 6
3. References . . . 7
4. Terms and definitions . . . 8
    4.1. Abbreviated terms . . . 8
5. Conventions . . . 10
    5.1. Identifiers . . . 10
    5.2. Examples . . . 10
    5.3. Schema representation . . . 10
    5.4. Use of HTTPS . . . 10
6. Introduction . . . 11
    6.1. Project Aims & Goals . . . 11
    6.2. Impact on users . . . 12
7. Core . . . 13
    7.1. Requirements Class "Core" . . . 13
    7.2. OpenAPI . . . 13
    7.3. Collection identifier . . . 13
    7.4. Collection title . . . 14
    7.5. Collection license . . . 14
    7.6. Collection temporal extent . . . 14
    7.7. Collection spatial extent . . . 14
    7.8. Data query response metadata . . . 15
    7.9. Collection vertical extent . . . 15
    7.10. Collection parameter names . . . 15
    7.11. Collection radius data query . . . 16
    7.12. Locations query response format . . . 16
    7.13. Error handling . . . 17
8. Insitu observations . . . 18
    8.1. Requirements Class "Insitu observations" . . . 18
    8.2. Collection data queries . . . 18
    8.3. Collection parameter names . . . 18
    8.4. Collection custom dimensions . . . 18
    8.5. Data query response format . . . 19
    8.6. CoverageJSON parameters . . . 19
    8.7. CoverageJSON referencing . . . 19
Annex A: Conformance Class Abstract Test Suite (Normative) . . . 21
    A.1. Conformance Class: Core . . . 21
        A.1.1. OpenAPI . . . 21

A.1.2. Collection identifier . . . 22
A.1.3. Collection title . . . 22
A.1.4. Collection license . . . 22
A.1.5. Collection spatial extent . . . 23
A.1.6. Collection temporal extent . . . 23
A.1.7. CoverageJSON referencing . . . 24
A.1.8. Collection parameter names . . . 24
A.1.9. Collection radius data query . . . 25
A.1.10. Locations query response format . . . 26
A.1.11. Error handling . . . 26
A.2. Conformance Class: Insitu observations . . . 27
A.2.1. Collection data queries . . . 27
A.2.2. Collection parameter names . . . 27
A.2.3. Collection custom dimensions . . . 28
A.2.4. Coveragejson parameters . . . 28
A.2.5. Data query response format . . . 29
Annex B: Schemas (Normative) . . . 30
Annex C: Examples (Informative) . . . 31
C.1. Examples . . . 31
Annex D: Bibliography . . . 32
Annex E: Revision History . . . 33

## i. Abstract

The Open Geospatial Consortium (OGC) Environmental Data Retrieval (EDR) API 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 - EDR identifies resources, captures compliance classes, and specifies requirements which are applicable to OGC Environmental Data Retrieval API's. The specification 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.

The flexibility of EDR allows for implementation in various environmental domains and communities of practice.

This document defines an EDR profile in support of providing access to meteorological datasets in the RODEO project. This includes, but is not limited to, conventions and constraints on identifiers, metadata parameters and encodings/formats and their related definitions and extensions.

## ii. Keywords

The following are keywords to be used by search engines and document catalogues.

rodeo, eumetnet, ogc, api, edr, weather, meteorology, high value datasets

## iii. Security Considerations

No additional security considerations have been made for this standard.

## iii. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

| Name | Affiliation |
| --- | --- |
| Håvard Futsæter (editor) | MET Norway |
| Tom Kralidis (editor) | Meteorological Service of Canada |

# Chapter 1. Scope

This document defines the conventions, constraints and extensions of EDR in the context of the RODEO project.

The RODEO EDR Profile defined herein is an extension of the International Standard *OGC API - Environmental Data Retrieval - Part 1: Core*.

This specification defines the conformance requirements for the RODEO EDR Profile. Annex A defines the abstract test suite. Annex B provides normative information on schemas. Annex C provides informative examples.

[1] https://rodeo-project.eu/partners/#project-partners

# Chapter 2. Conformance

Conformance with this standard shall be checked using the tests specified in Annex A (normative) of this document.

*OGC API - Environmental Data Retrieval* (EDR) provides a family of lightweight interfaces to access Environmental Data resources. Each resource addressed by an EDR API maps to a defined query pattern. This standard is a profile/extension of EDR. Conformance to this standard requires demonstrated conformance to the applicable Conformance Classes of *OGC API - Environmental Data Retrieval - Part 1: Core*.

Implementors of EDR API within the RODEO project are required to comply with the RODEO EDR Profile. A RODEO EDR API shall therefore be compliant with OGC API - Environmental Data Retrieval - Part 1: Core.

This standard identifies the following Requirements Classes which define the functional requirements.

- "Core": baseline requirements class required by all other requirements classes in this document
- "Observations": TBD
- "Radar": TBD

Compliant implementation of this profile requires conformance to at least the "Core" Requirements Class.

# Chapter 3. References


- OGC: OGC 19-086r6, OGC API - Environmental Data Retrieval Standard - Part 1: Core (2023) [1]
- OGC: OGC 21-069r2, OGC CoverageJSON Community Standard (2023) [2]
- IETF: RFC-7946 The GeoJSON Format (2016) [3]
- IETF: RFC-8259 The JavaScript Object Notation (JSON) Data Interchange Format (2017) [4]
- W3C/OGC: Spatial Data on the Web Best Practices, W3C Working Group Note (2017) [5]
- W3C: Data on the Web Best Practices, W3C Recommendation (2017) [6]
- IANA: Link Relation Types (2020) [7]
- IANA: Media Types (2023) [8]
- IETF: JSON Schema (2022) [9]
- OpenAPI Specification 3.1.0 (2022) [10]


[1] https://docs.ogc.org/is/19-086r6/19-086r6.html

[2] https://docs.ogc.org/cs/21-069r2/21-069r2.html

[3] https://datatracker.ietf.org/doc/html/rfc7946

[4] https://datatracker.ietf.org/doc/html/rfc8259

[5] https://www.w3.org/TR/sdw-bp

[6] https://www.w3.org/TR/dwbp

[7] https://www.iana.org/assignments/link-relations/link-relations.xml

[8] https://www.iana.org/assignments/media-types/media-types.xhtml

[9] https://json-schema.org

[10] https://github.com/OAI/OpenAPI-Specification/blob/3.1.0/versions/3.1.0.md

# Chapter 4. Terms and definitions

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “SHALL” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this Standard and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the 'ModSpec'. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

The following additional terms and definitions also apply.

## 4.1. Abbreviated terms

*Table 1. Symbols and abbreviated terms*

| Abbreviation | Term |
|---|---|
| API | Application Programming Interface |
| EU | European Union |
| EUMETNET | European National Meteorological and Hydrological Services |
| FAIR | Findable, Accessible, Interoperable, Reusable |
| FEMDI | Federated European Meteo-hydrological Data Infrastructure |
| GIS | Geographic Information System |
| HVD | High Value Datasets |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| IANA | Internet Assigned Numbers Authority |
| IETF | Internet Engineering Task Force |
| ISO | International Organization for Standardization |
| JSON | JavaScript Object Notation |
| MIME | Multipurpose Internet Mail Extensions |
| NMHS | National Meteorological and Hydrological Service |
| NWP | Numerical Weather Prediction |
| OGC | Open Geospatial Consortium |

| Abbreviation | Term |
| --- | --- |
| REST | Representational State Transfer |
| ROA | Resource-oriented architecture |
| RODEO | The provision of open access to public meteorological data and development of shared federated data infrastructure for the development of information products and services |
| S3 | Simple Storage Service |
| SEO | Search engine optimization |
| SOA | Service-oriented architecture |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| W3C | World Wide Web Consortium |
| XML | eXtensible Markup Language |

# Chapter 5. Conventions

This section provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of JSON and / or JSON schema, or special notes regarding how to read the document.

## 5.1. Identifiers

The normative provisions in this Standard are denoted by the URI:

https://rodeo-project.eu/spec/rodeo-edr-profile/1

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

## 5.2. Examples

Examples provided in this specification are encoded as JSON, GeoJSON or CoverageJSON.

## 5.3. Schema representation

JSON Schema [1] objects are used throughout this standard to define the structure of metadata records. These schema objects are also typically represented using YAML [2]. YAML is a superset of JSON, and in this standard are regarded as equivalent.

## 5.4. Use of HTTPS

For simplicity, this document only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS and simply is a shorthand notation for "HTTP or HTTPS." In fact, most servers are expected to use HTTPS, not HTTP.

[1] https://json-schema.org

[2] https://en.wikipedia.org/wiki/YAML

# Chapter 6. Introduction

RODEO, the Provision of Open Access to Public Meteorological Data and Development of Shared Federated Data Infrastructure for the Development of Information Products and Services, is a joint effort by eleven European National Meteorological and Hydrological Services (NMHS), the European Centre for Medium-Range Weather Forecasts (ECMWF) and the network of 31 European National Meteorological and Hydrological Services (EUMETNET). The Project Partners are listed in the page Partners [1].

The RODEO project develops a user interface and Application Programming Interfaces (API) for accessing meteorological datasets declared as High Value Datasets (HVD) by the EU Implementing Regulation (EU) 2023/138 under the EU Open Data Directive (EU) 2019/1024. The project also fosters the engagement between data providers and data users for enhancing the understanding of technical solutions being available for sharing and accessing the HVD datasets.

Surface weather observations, climate data, weather warnings, radar data and numerical weather prediction (NWP) data are defined as meteorological high value datasets. The distribution of these datasets is going to be made available under an open licence, in machine-readable formats using APIs and bulk downloads following the FAIR principles (usability, accessibility, interoperability and reusability).

The project is funded by the EU Digital Europe Program (DIGITAL) and EUMETNET.

## 6.1. Project Aims & Goals

The project makes meteorological High Value Datasets easily available with an aim to bring new data to businesses, public administrations and citizens. It reinforces the European public meteorological data infrastructure and enhances the providers' digital capacity to share data. Furthermore, the project also fosters the engagement between data providers and data users for enhancing the understanding of technical solutions to be available for sharing and accessing the datasets.

The project strengthens the capacity of the European meteorological data providers to comply with the HVD Implementing Regulation by:

- Developing a user interface
- Developing APIs for accessing weather observation data, climate data, weather radar data, warnings, and Artificial Intelligence (AI) datasets
- Developing a data catalogue for making data discoverable
- Engaging with the data owners and user communities
- Supporting the deployment of national data portals and APIs
- Making HVDs available from the project partners

## 6.2. Impact on users

The increased availability of data boosts entrepreneurship and potentially results in the creation of new companies. New open datasets are an important resource for small and medium-sized enterprises to develop new digital products and services. Reuse of data opens business opportunities for several sectors ultimately leads attracting more investors. By making data much easier to use, reseach, academia, AI applications and many other application areas can utilize the data more efficiently.

Overall, better data availability leads to better warnings, forecasts, and services to the public and weather-critical industries, and contributes to the safe and efficient functioning of society with multiple benefits across the European economy, industry, and society.

[1] https://eurodeo.github.io/partners

# Chapter 7. Core

## 7.1. Requirements Class "Core"

| | |
|---|---|
| URI | https://rodeo-project.eu/spec/rodeo-edr-profile/1/req/core |
| Target type | Web API |
| Dependency | OGC API - Environmental Data Retrieval Standard - Part 1: Core (2023) [1] |
| Pre-conditions | The API conforms to OGC API - Environmental Data Retrieval - Part 1: Core, Requirements Class: Core and Requirements Class: Collections |

## 7.2. OpenAPI

An OpenAPI specification provides a machine-readable description of the API interface.

| **Requirement 1** | **/req/core/openapi** |
|---|---|
| A | An API definition SHALL be described using an OpenAPI document (version 3.1 or higher). |
| B | The OpenAPI document SHALL be encoded as JSON. |
| C | The OpenAPI document SHALL be made available in the API Landing Page as a link object with a relation type of `service-desc`. |
| D | API documentation SHALL be made available in the API Landing Page as a link object with a relation type of `service-doc` and content type of `text/html`. |

## 7.3. Collection identifier

A collection identifier provides a mechanism to uniquely identify a given collection in an OGC API.

| **Requirement 2** | **/req/core/collection_identifier** |
|---|---|
| A | A collection identifier SHALL NOT be used to convey structured metadata. |
| B | A collection identifier SHOULD use the following list of values, each suitable for a specific data type: insitu-observations, climate_data, radar_observations, weather_warnings, weather_forecast. |
| C | The identifier string MAY if needed include a postfix to the values listed above. |

## 7.4. Collection title

A collection title provides a human readable short description of the given collection.

| Requirement 3 | /req/core/collection_title |
|---|---|
| A | A title SHALL be set for all collections. |
| B | A title SHALL NOT have more than 50 characters. |
| C | A title SHOULD be written in English. |
| D | A title SHOULD have the most important information first, guarding against truncated presentation of the value. |
| E | A title SHOULD describe the collection in a way understandable also for non-experts. Usually mention both topic/domain and geographical area. |

## 7.5. Collection license

A license describes the usage permissions for the data in a collection.

| Requirement 4 | /req/core/collection_license |
|---|---|
| A | The `links` property in a `collection` SHALL contain a link to a license of its data. |
| B | The license link SHALL set the `rel` property to `license`. |
| C | The `type` property SHALL be set to `text/html`. |
| D | If the data is open data, the license SHALL be `CC BY 4.0` and the `href` property set to `https://creativecommons.org/licenses/by/4.0/`. |

## 7.6. Collection temporal extent

Temporal extent of a collction.

| Requirement 5 | /req/core/collection_temporal_extent |
|---|---|
| A | The value of `extent.temporal.trs` SHALL be `Gregorian`. |

## 7.7. Collection spatial extent

Spatial extent of a collction.

| Requirement 6 | /req/core/collection_spatial_extent |
|---|---|
| A | The value of `extent.spatial.crs` SHALL be `OGC:CRS84`. |
| B | The value of `crs` SHALL include the element `OGC:CRS84`. |

| | |
|---|---|
| C | If `crs_details` is specified for a `data_queries` object, the `crs_details` array SHALL include an object whose `crs` value is `OGC:CRS84`. |
| D | If a collection supports more CRSs than `OGC:CRS84`, these values SHOULD be presented on the syntax `EPSG:<code>` in `crs` and `data_queries.*.crs_details.crs` if an epsg code is available. |

## 7.8. Data query response metadata

Metadata in the response to a data query.

| **Recommendation 1** | **/rec/core/data_query_response_metadata** |
|---|---|
| A | The metadata in the response to a data query, e.g crs, parameters and units, SHOULD mirror as closely as possible the terms and vocabularies used in the collection metadata. |

## 7.9. Collection vertical extent

Vertical extent of a collection.

| **Recommendation 2** | **/rec/core/collection_vertical_extent** |
|---|---|
| A | If `extent.vertical` is defined, the value of `extent.vertical.vrs` SHOULD be one the following values: `Pressure level in hPa`, `Geopotential height in gpm`, `Geometrical altitude above mean sea level in meters`, `Height above ground in meters`, `Flight level`. |

## 7.10. Collection parameter names

Parameter names is an object, containing parameter objects with metadata about all parameters the collection contains.

| **Requirement 7** | **/req/core/collection_parameter_names** |
|---|---|
| A | The keys in `parameter_names` and `id` in the parameter object SHALL NOT be used to convey structured metadata. |
| B | A `parameter` object SHALL specify the following fields: `label`, `description`, `unit` |
| C | `label` SHALL NOT have more than 50 characters. |
| D | `label` SHALL be written in English. |
| E | `unit` SHALL use QUDT as vocabulary, and `unit.symbol.type` SHALL be on the form `https://qudt.org/vocab/unit/<unit>` and `unit.symbol.value` SHALL be set to the value of `qudt:symbol` for that unit. |

| | |
|---|---|
| F | `observedProperty` SHALL use the CF-convention if a suitable value is available and set `observedProperty.id` to value `https://vocab.nerc.ac.uk/standard_name/<standard_name>` If not, use `observedProperty.description` to describe it. |

## 7.11. Collection radius data query

Metadata about the radius data query in a collection.

| **Requirement 8** | **/req/core/collection_radius_data_query** |
|---|---|
| A | The `within_units` array in a `variables` object for a radius data query SHALL at least contain `m`, interpreted as the SI unit of length. |
| B | `within_units` can contain more values, but all values in the array SHALL be defined as a unit https://qudt.org, and SHALL use the symbol of the unit. |
| C | When performing a radius query which includes the `limit` query parameter, this SHOULD be interpreted as requesting the limited amount of data items that are closest to the specified center of the radius. |

## 7.12. Locations query response format

Structure of the response document for a /locations query.

| **Requirement 9** | **/req/core/locations_query_response_format** |
|---|---|
| A | The response document for a `/locations` query to a `collection` or an `instance` of a `collection` SHALL be a GeoJSON document with a FeatureCollection object. |
| B | Each Feature object in the FeatureCollection SHALL have the following properties: `id` of type string, `properties.name` of type string. |
| C | If the data for a location do not provide all parameters listed in the `collection`, it SHALL include `properties.parameter-name` which SHALL be a list of strings. This list SHALL be a subset of the list of keys from the `parameter_names` object in the collection. |
| D | If there are links with more information about the location they SHALL be included in `properties.links` as a list of `link` objects. The `link` object SHALL have identical structure as link objects defined by OGC API - Common Part 1: Core. |
| E | A more detailed description of the location MAY also be provided via `properties.description`. |
| F | All strings SHALL be in english. |

## 7.13. Error handling

HTTP status codes and response documents for responding to errors.

| Requirement 10 | /req/core/error_handling |
|---|---|
| A | A query for data inside the extent of a collection that results in no data SHALL have a HTTP status code of `204`, and return no content. |
| B | A data query that is partially within the extent SHALL respond as if it is within the extent, but SHOULD only respond with data for the part of the geographical area in the query that is inside the extent. |
| C | Any http request that results in a response with a 4xx HTTP status code SHALL give a response which complies with RFC 9457 Problem Details for HTTP API. |

[1] https://docs.ogc.org/is/19-086r6/19-086r6.html

# Chapter 8. Insitu observations

## 8.1. Requirements Class "Insitu observations"

| | |
|---|---|
| URI | https://rodeo-project.eu/spec/rodeo-edr-profile/1/req/insitu-observations |
| Target type | Web API |
| Dependency | Core |

## 8.2. Collection data queries

The types of data queries implemented in a collection.

| **Requirement 11** | **/req/insitu-observations/collection_data_queries** |
|---|---|
| A | A collection SHALL support locations, area and radius. |

## 8.3. Collection parameter names

Parameter names is an object, containing parameter objects with metadata about all parameters the collection contains.

| **Requirement 12** | **/req/insitu-observations/collection_parameter_names** |
|---|---|
| A | Each object in `parameter_names` SHALL specify a property `metocean:standard_name` with value set to one of the values listed in `values` array for custom dimension with `id` with value `standard_names`. |
| B | Each object in `parameter_names` SHALL specify a property `metocean:level` with value set to one of the values listed in `values` array for custom dimension with `id` with value `levels`. |
| C | Each object in `parameter_names` SHALL specify `measurementType` and the `measurementType.method` SHOULD follow the CF-convention cell methods https://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/build/ch07s03.html and use a value from the list in https://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/build/ape.html. |

## 8.4. Collection custom dimensions

The custom dimensions for which you can query the collection.

| **Requirement 13** | **/req/insitu-observations/collection_custom_dimensions** |
|---|---|

| | |
|---|---|
| A | A collection can contain multiple parameters who fit under a standard name as defined by the CF metadata conventions. A collection SHALL have an object in `extent.custom` with `id` of value `standard_name`. |
| B | Custom dimension object `standard_name` SHALL specify a list of CF-conventions standard names in `values`. |
| C | Custom dimension `standard_name` SHALL specify `reference` with value `https://vocab.nerc.ac.uk/standard_name/`. |
| D | A collection can contain parameters that are identical except for having performed measurements on different heights. A collection SHALL have an object in `extent.custom` with `id` of value `level`. |
| E | Custom dimension `level` SHALL specify `reference` with value `Height of measurement above ground level in meters`. |

## 8.5. Data query response format

The response format for data queries.

| **Requirement 14** | **/req/insitu-observations/data_query_response_format** |
|---|---|
| A | All data queries SHALL support at least CoverageJSON as response format. |

## 8.6. CoverageJSON parameters

The metadata about parameters in CoverageJSON.

| **Requirement 15** | **/req/insitu-observations/coveragejson_parameters** |
|---|---|
| A | Each object in `parameters` in a `coverage` object SHALL have a property `metocean:measurementType`. This object SHALL have the same properties as `measurementType` in an object in `parameter_names` in a `collection`. |
| B | Each object in `parameters` SHALL have a property `metocean:standard_name`. |
| C | Each object in `parameters` SHALL have a property `metocean:level`. |

## 8.7. CoverageJSON referencing

Coordinate referencing metadata in CoverageJSON.

| **Requirement 16** | **/req/insitu-observations/coveragejson_referencing** |
|---|---|

| | |
|---|---|
| A | When requesting data and crs query param is not specified the `system` property for the spatial coordinates in `domain.referencing.*` SHALL have the following values: `"type": "GeographicCRS"` and `"id": "OGC:CRS84"`. |
| B | When requesting data and the crs query param is specified, the `domain.referencing.*.system` object with `"type": "GeographicCRS"` SHALL have its `id` property set to the same value as the crs query param. |

# Annex A: Conformance Class Abstract Test Suite (Normative)

## A.1. Conformance Class: Core

**label**

https://rodeo-project.eu/spec/rodeo-edr-profile/1/conf/core

**subject**

Requirements Class "core"

**classification**

Target Type:Web API

### A.1.1. OpenAPI

**label**

/conf/core/openapi

**subject**

/req/core/openapi

**test-purpose**

Validate that a RODEO EDR Profile provides an API definition using an OpenAPI document

Construct a path for a landing page.

Issue an HTTP GET request on that path.

Check that the value of the returned HTTP status header is 200.

In the `links` array in the landing page response, find the object with `rel` set to `service-desc` and `type` set to `application/vnd.oai.openapi+json;version=3.1`.

Issue an HTTP GET request using the value of `href` for that object.

Check that the value of the returned HTTP status header is 200 and check that the response body is a valid OpenAPI document of version 3.1 or higher.

In the `links` array in the landing page response, find the object with `rel` set to `service-doc`.

Issue an HTTP GET request using the value of `href` for that object.

Check that the value of the returned HTTP status header is 200 and that the HTTP response header `Content-Type` contains `text/html`.

### A.1.2. Collection identifier

**label**

/conf/core/collection_identifier

**subject**

/req/core/collection_identifier

**test-purpose**

Validate that a RODEO EDR Profile API provides a valid collection identifier

This requirement is not applicable to ATS testing.

### A.1.3. Collection title

**label**

/conf/core/collection_title

**subject**

/req/core/collection_title

**test-purpose**

Validate that a RODEO EDR Profile provides a valid collection title

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the `collections` array, check that the `title` property is present for all array elements.

For each `title` value, check that the value is less than or equal to 50 characters.

### A.1.4. Collection license

**label**

/conf/core/collection_license

**subject**

/req/core/collection_license

**test-purpose**

Validate that a RODEO EDR Profile API provides a collection links array with a license link.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the `links` array, check that there exists one `link` object with `rel` set to `license`.

Issue a HTTP GET request to the url in `href` for `link` with rel `license`.

Check that the value of the returned HTTP status header is 200 and the HTTP response header `Content-Type` is `text/html`.

### A.1.5. Collection spatial extent

**label**

/conf/core/collection_spatial_extent

**subject**

/req/core/collection_spatial_extent

**test-purpose**

Validate that a RODEO EDR Profile API provides spatial extent in the correct crs.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the response document, for each object in the `collections` array, check that the `extent.spatial.crs` property is set to `OGC:CRS84`.

In the `collections` array, check that the `crs` property is set to `OGC:CRS84` for each object.

For each object in `collections` array, for each object in the `data_queries` array, if `crs_details` is specified, check that one of the objects in the `crs_details` array has a `crs` property set to `OGC:CRS84`.

### A.1.6. Collection temporal extent

**label**

/conf/core/collection_temporal_extent

**subject**

/req/core/collection_temporal_extent

**test-purpose**

Validate that a RODEO EDR Profile API provides temporal extent in the correct trs.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the response document, for each object in the `collections` array, check that the `extent.temporal.trs` property is set to `Gregorian`.

### A.1.7. CoverageJSON referencing

**label**

/conf/core/coveragejson_referencing

**subject**

/req/core/coveragejson_referencing

**test-purpose**

Validate that a RODEO EDR Profile API CoverageJSON response document provides correct domain referencing values.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

For each object in the `collections` array, for each object in `data_queries`, issue a GET request with no crs query parameter specified.

Check that the response is a CoverageJSON document where each object of `type Coverage` contains a `domain.referencing` array which includes an object where `system.type` is set to `GeographicCRS` and `system.id` set to `OGC:CRS84`.

For each object in the `collections` array, for each object in `data_queries`, issue a GET request for each `crs_details` object where `crs` query parameter set to the value in the `crs_details` object.

Check that the response is a CoverageJSON object where `domain.referencing` includes an object where `system.type` is set to `GeographicCRS` and `system.id` is equal to the crs query parameter.

### A.1.8. Collection parameter names

**label**

/conf/core/collection_parameter_names

**subject**

/req/core/collection_parameter_names

**test-purpose**

Validate that a RODEO EDR Profile API provides a valid collection parameter_names

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the response document, for each object in the `collections` array, check that each object in the `parameter_names` array has the properties `label`, `unit` and `description`.

The value of a `label` property in a `parameter` object is less than or equal to 50 characters.

The value of `unit.symbol.type` in a `parameter` object is a url on the form `https://qudt.org/vocab/unit/{unit}`.

The value of `unit.symbol.value` in a `parameter` object is specified.

### A.1.9. Collection radius data query

**label**

/conf/core/collection_radius_data_query

**subject**

/req/core/collection_radius_data_query

**test-purpose**

Validate that a RODEO EDR Profile API with a radius data query in a collection has correct metadata.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the response document, for each object in the `collections` array, if the `data_queries` property contains `radius`, check that the `link.variables` object for `radius` has a `within_units` array that contains at least `m`.

### A.1.10. Locations query response format

**label**

/conf/core/locations_query_response_format

**subject**

/req/core/locations_query_response_format

**test-purpose**

Validate that a RODEO EDR Profile API has a correctly formated response to a locations query.

For every collection and instance of a collection, check if `locations` exists as a key in the `data_queries` object. If so, get the `href` value from the link object with `"rel":"self"` and append `/locations` to the value and issue an HTTP GET request.

Check that the value of the returned HTTP status header is 200.

Check that the response http header `Content-Type` has value `application/geo+json`.

In the response document, check that it has the property `"type": "FeatureCollection"`. For each object in the `features` array, check that the properties `id` and `properties.name` are present.

### A.1.11. Error handling

**label**

/conf/core/error_handling

**subject**

/req/core/error_handling

**test-purpose**

Validate that a RODEO EDR Profile API handles errors correctly.

Issue an HTTP GET request to an non-existent path `/collections/nonexistentcollection`.

Check that the value of the returned HTTP status code is `404` and the HTTP response header `Content-Type` is `application/problem+json`.

Check that the returned JSON document includes the properties `type` and `title`. Check that `title` is a string and that `type` is either an URI or the value `about:blank`.

# A.2. Conformance Class: Insitu observations

**label**

https://rodeo-project.eu/spec/rodeo-edr-profile/1/conf/insitu-observations

**subject**

Requirements Class "Insitu observations"

**classification**

Target Type:Web API

## A.2.1. Collection data queries

**label**

/conf/insitu-observations/collection_data_queries

**subject**

/req/insitu-observations/collection_data_queries

**test-purpose**

Validate that a RODEO EDR Insitu observations Profile API has implemented the mandatory data queries.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the `collections` array in the returned document, check that each collection has a `data_queries` array containing at least `area`, `locations` and `radius`.

## A.2.2. Collection parameter names

**label**

/conf/insitu-observations/collection_parameter_names

**subject**

/req/insitu-observations/collection_parameter_names

**test-purpose**

Validate that a RODEO EDR Insitu observations Profile API has the required properties for parameter_names

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

For each object in the `collections` array and for each object in `parameter_names` check that it has the properties `metocean:standard_name` with the value of type string.

For each object in the `collections` array and for each object in `parameter_names` check that it has the property `metocean:level` with a value of type number.

For each object in the `collections` array and for each object in `parameter_names` check that is has the property `measurementType`.

### A.2.3. Collection custom dimensions

**label**

/conf/insitu-observations/collection_custom_dimensions

**subject**

/req/insitu-observations/collection_custom_dimensions

**test-purpose**

Validate that a RODEO EDR Insitu observations Profile API has required custom dimensions.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

Check that the `extent.custom` array has an object with property `id` set to `standard_name` and property `reference` set to `https://vocab.nerc.ac.uk/standard_name/`.

Check that the `extent.custom` array has an object with property `id` set to `level` and property `reference` set to `Height of measurement above ground level in meters`.

### A.2.4. Coveragejson parameters

**label**

/conf/insitu-observations/coveragejson_parameters

**subject**

/req/insitu-observations/coveragejson_parameters

**test-purpose**

Validate that a RODEO EDR Insitu observations Profile API has required metadata in coveragejson parameters.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

For each object in the `collections` array and for each object in `data_queries`, issue a GET request with no crs query parameter specified.

Check that each object in `parameters` array of the response CoverageJSON document contains a property `metocean:standard_name` with a string type.

Check that each `parameter` object in `parameters` in the response CoverageJSON document contains a property `metocean:level` with a number type.

Check that each `parameter` object in `parameters` in the response CoverageJSON document contains a property `metocean:measurementType` which has an object type. Check that this object has the property `method` of type string and property `duration` of type string.

### A.2.5. Data query response format

**label**

/conf/insitu-observations/data_query_response_format

**subject**

/req/insitu-observations/data_query_response_format

**test-purpose**

Validate that a RODEO EDR Insitu observations Profile API has correct response format for data queries.

Issue an HTTP GET request to path `/collections`.

Check that the value of the returned HTTP status header is 200.

In the returned document, for each object in `data_queries` for each collection in `collections` make a corresponding data query request and check that the returned response has status 200 and has a http response header `Content-Type` with value `application/vnd.cov+json`.

# Annex B: Schemas (Normative)

# Annex C: Examples (Informative)

## C.1. Examples

# Annex D: Bibliography


- W3C/OGC: Spatial Data on the Web Best Practices, W3C Working Group Note 28 September 2017, https://www.w3.org/TR/sdw-bp
- W3C: Data on the Web Best Practices, W3C Recommendation 31 January 2017, https://www.w3.org/TR/dwbp
- W3C: Data Catalog Vocabulary, W3C Recommendation 16 January 2014, https://www.w3.org/TR/vocab-dcat
- IANA: Link Relation Types, https://www.iana.org/assignments/link-relations/link-relations.xml
- Linux Foundation: SPDX License List, https://spdx.org/licenses

# Annex E: Revision History

| Date | Release | Editor | Primary clauses modified | Description |
|---|---|---|---|---|
| 2024-09-16 | Template | Tom Kralidis | all | initial template |