

OGC® DOCUMENT: 23-047

External identifier of this OGC® document: <http://www.opengis.net/doc/PER/T19-D011>



Open
Geospatial
Consortium

OGC TESTBED-19 GEODATACUBES ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2024-03-05

Approval Date: 2024-03-27

Publication Date: 2024-07-22

Editor: Alexander Jacob

Notice: This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is *not an official position* of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

License Agreement

Use of this document is subject to the license agreement at <https://www.ogc.org/license>

Copyright notice

Copyright © 2024 Open Geospatial Consortium
To obtain additional rights of use, visit <https://www.ogc.org/legal>

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

I.	EXECUTIVE SUMMARY	vi
II.	KEYWORDS	vii
III.	CONTRIBUTORS	vii
2.	INTRODUCTION	10
3.	STATE OF THE ART	14
3.1.	Crosswalk between STAC and OGC API – Records	14
3.2.	Crosswalk between the openEO API specification and OGC API – Processes Standard	23
3.3.	Crosswalk between the openEO API specification and the draft OGC API – Coverages Standard	32
3.4.	Crosswalk between the draft GDC standard and the OGC WCPS Standard	41
4.	HIGH LEVEL OVERVIEW OF GDC API	45
4.1.	Profiles proposal by Ecere	46
5.	OUTLINE OF ALL CONDUCTED EXPERIMENTS/IMPLEMENTATIONS	49
5.1.	Back-ends	49
5.2.	Clients	72
6.	INTER COMPARISON EXPERIMENTS	99
6.1.	Ecere Technology Integration Experiments	99
6.2.	Eurac Research – GDC Web Editor	105
6.3.	Geomatys – API endpoints integration	106
6.4.	Wuhan University Technology Integration Experiments	108
6.5.	52°North GmbH – Executable Test Suite (ETS) for the draft OGC API – GDC Standard	109
7.	USABILITY TESTS	111
7.1.	Sinergise Usability Test	111
8.	LESSONS LEARNED FROM API IMPLEMENTATION	116
9.	FUTURE OUTLOOK	119
	ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS	121
	BIBLIOGRAPHY	124

LIST OF TABLES

Table 1	15
Table 2	16
Table 3	16
Table 4	18
Table 5	19
Table 6	42
Table 7	50
Table 8 – Use case A (mapping of capabilities)	53
Table 9 – Use case B (mapping of capabilities)	53
Table 10 – Use case C (mapping of capabilities)	54
Table 11 – Status of the GNOSIS Cartographer and/or gdc-test client integration with participant API endpoints	99
Table 12	106
Table 13	106
Table 14	108
Table 15	109
Table 16	113

LIST OF FIGURES

Figure 1	42
Figure 2	42
Figure 3 – GDC Draft Specification High Level Overview	46
Figure 4 – OGC API - Coverages request from GNOSIS Map Server using CQL2 expressions to filter cells by values and convert Kelvin to Celsius	55
Figure 5 – Coverage output of above request, with a color map style applied in QGIS	55
Figure 6 – OGC API - Coverages request from GNOSIS Map Server using CQL2 expressions to compute an Enhanced Vegetation Index (EVI) and filter out clouds	55
Figure 7 – Coverage output of above request, with a color map styled applied in QGIS	56
Figure 8 – Example PassThrough process execution request (for Collection Output)	56
Figure 9 – Example PassThrough process execution request (for Synchronous execution)	57
Figure 10 – Filtering using CQL2 polygon geometry	58
Figure 11 – Aggregating on temporal dimension using CQL2 expression	58
Figure 12 – Sobel operator (kernel convolution) implemented as a CQL2 expression	58
Figure 13 – OGC API - Coverages request from Eurac Research client (GDC Web Editor) to our server, visualized as an RGB composite	61
Figure 14	62

Figure 15	70
Figure 16	71
Figure 17 – Cartographer visualizing CMIP5 pressure and wind velocity	73
Figure 18 – Cartographer visualizing relative humidity	74
Figure 19	76
Figure 20	77
Figure 21	78
Figure 22	78
Figure 23 – WHU’s OGE-DataClient visualizing climate:era5:relativeHumidity from Ecere Coverages API	87
Figure 24 – WHU’s OGE-DataClient visualizing processing result from Ecere Processes API using synchronous mode	87
Figure 25 – WHU’s OGE-DataClient visualizing processing result from WHU Processes API using Workflows&Chaining(collection output)	88
Figure 26 – WHU’s OGE-DataClient visualizing processing result in PNG format obtained from the rasdaman OpenEO API while operating in synchronous mode	89
Figure 27 – Albedo as an Essential Variable of the Arctic	91
Figure 28 – EDR provider framework	92
Figure 29 – Surface albedo Spring 2019	93
Figure 30 – CMIP5 Near Surface Temperature Spring 2019	93
Figure 31 – Magnitude of change in surface albedo between May 2019 and April 2023	94
Figure 32 – Tree Cover Density dataset from rasdaman’s GDC API visualized in 3D using GNOSIS Cartographer	101
Figure 33 – EVAPOTRANSPIRATION dataset from Brockmann Consult retrieved from Ecere’s gdc-test client (styled in QGIS)	102
Figure 34 – Successful coverage request from Wuhan University server ECMWF_hsvs collection (styled in QGIS)	103
Figure 35 – Successful coverage request from Eurac server s2_l2a collection	104
Figure 36 – Ecere’s GDC Test tool showing syntax help for command line arguments	105
Figure 37 – Ecere’s GDC Test tool executed for the GNOSIS Map Server endpoint	105



EXECUTIVE SUMMARY

OGC Testbed-19 has continued and furthered an ongoing discussion about how to interact with GeoDataCubes (GDC) in the most interoperable way (see Chapter 1 for more Introduction). Testbed 19 participants produced a draft OGC GDC API standard that incorporates the most relevant developments in the field in and outside of OGC. This work advanced the common understanding of available solutions while discovering to a much better degree the advantages and drawbacks of current solutions. Testbed 19 participants produced prototypes of five back-end implementations and six client implementations as well as an automated test suite, which are described in full detail in Chapter 4. Many of the researched solutions are also available as open source and hence offer a perfect starting point for further GDC activities.

The main technologies that were evaluated in Testbed 19 included the OGC API Standards suite¹, the openEO API² and the Spatiotemporal Asset Catalog³ (STAC) specification. Based on cross walk comparisons (see Chapter 2), a unified draft GDC API was developed integrating as much as possible the existing solutions. openEO is largely compliant with the OGC API-Common Standard. As such, the openEO API specification provided the foundation for defining a draft OGC GDC API draft standard. During the Testbed 19 period, more building blocks from the OGC API were incorporated into the draft GDC API document. These building blocks included parts of OGC API – Common, OGC API – Coverages, and OGC API – Processes. There is also future potential for visualization services through maps or tiles or even including components or elements of the OGC Web Services suite of Standards, such as WMS, WMTS, WCS, etc.

The current version of the draft GDC API, described in D71 of T19, supports different scenarios enabling implementations of the draft standard to offer only minimal support for data access with minimal manipulation of the data. Minimal manipulation is in terms of subsetting and reprojecting or including more advanced processing capabilities by incorporating building blocks from the openEO specification or from the OGC API – Processes – Part 1: Core Standard. Chapter 3 gives an overview of the draft standard.

The interaction capabilities between the different servers and clients developed are described in Chapter 5 and first impressions on usability in Chapter 6.

Future work could include the ability to link two processing options into one “integrated” option that supports either submitting openEO process graphs to a OGC API – Processes endpoint (extending and working on Processes – Part 3), or supports integration of an OGC API – Processes process in the process graph of openEO through an extended concept of user defined functions in openEO. Further discussion is also needed on the pros and cons of including

¹<https://ogcapi.ogc.org/>

²<https://openeo.org/>

³<https://stacspec.org/en>

authentication in the draft standard. More details about lessons learned and suggestions can be found in Chapters 7 and 8 of this ER.



KEYWORDS

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

geographic, data cubes, api



CONTRIBUTORS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Alexander Jacob	Eurac Research	Editor
Matthias Mohr	Eurac Research	Contributor
Michele Claus	Eurac Research	Contributor
Glenn Laughlin	Pelagis Data Solutions	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor
Patrick Dion	Ecere Corporation	Contributor
Dimitar Misev	Rasdaman	Contributor
Peter Baumann	Rasdaman	Contributor
Quentin Bialota	Geomatys	Contributor
Pascal Broglie	Geomatys	Contributor
Gérald Fenoy	GeoLabs	Contributor
Peng Yue	Wuhan University	Contributor

NAME	ORGANIZATION	ROLE
Kaixuan Wang	Wuhan University	Contributor
Yi Wei	Wuhan University	Contributor
Mahael Kadunc	Planet Labs	Contributor
Benjamin Pross	52° North	Contributor
Chen-Yu Hao	Feng Chia University	Contributor
Pontus Lurcock	Brockmann Consult	Contributor
Greg Buehler	Open Geospatial Consortium	Initial Template



2

INTRODUCTION

In order to reduce the time and effort required to generate added value from raw, heterogeneous data, over the past decade, a multitude of independent initiatives have developed solutions to answer the need for Analysis Ready Data (See also Testbed 19 ARD ER D051). These initiatives resulted in various GeoDataCube (GDC) implementations, standards, data formats, and best practices. Interoperability between the GDC solutions has so far not been a core concern. However, with the increasing amount of data served as a GDC as well as its increasing uptake, it is becoming essential to understand what exactly a GDC entails, how it was created, and how different GDCs can be consistently used together. This iteration of GDC and ARD concepts in OGC Testbeds has started researching into the current landscape of GDC technology. Based on an evaluation of existing solutions and applications, the Testbed participants designed and implemented a first draft version of a GDC API that specifies requirements for supporting data access and processing within and across multiple instances of GDCs. This draft GDC API standard provides the core functionalities of GDCs in general and enables users to handle different GDC implementations/back-ends according to the same principles. By improving interoperability, both vendors and consumers will benefit. At the same time, vendors and data providers will be able to continue the development of specific GDC variants with full flexibility regarding design and technological choices (e.g., internal storage organization) as the GDC focuses solely on ensuring interoperable interfaces. On the other hand, data consumers and end users will be able to interact more effectively with different GDC instances. This can be achieved by merging, combining, and creating workflows spanning across multiple GDCs, easily migrating between different solutions while being protected from vendor lock-in. For the Testbed 19 GDC activity, use cases were prepared by several stakeholders (NRCan, EUMETSAT, ECMWF). Below is a short description of these selected use cases:

- NRCan's use case focused on evaluating higher-level GDC requirements in the context of emerging Canadian activities related to improving the use of Earth Observation (EO) nationally. NRCan wanted to incorporate developed GDC API concepts within the design of future Canadian EO exploitation platforms to maximize the platforms' utility and interoperability. In this context, it will be beneficial for the OGC GDC API Standard to support:
 - access to diverse EO data sets from domestic and international sources;
 - integration of EO and other forms of geospatial data (e.g., vector) for analysis purposes;
 - access to dataset and GDC metadata to support diverse use of the GDC and the data it contains (e.g., for analysis, processing, etc.);
 - interoperability with GDCs offered by multiple providers; and
 - implementation within open source software packages.
- The EUMETSAT use case draws inspiration from the services to be provided by the Destination Earth Data Lake project. A key challenge in the Data Lake project is the heterogeneity of data types that must be provided to users in a harmonized manner, making the data easily exploitable. The proposed GeoDataCube contains weather

and climate data, administrative and environmental spatial units (vector maps), static environmental data, land use/land cover maps, and remote sensing image time series. Users exploiting the GeoDataCube must be able to query external statistical databases. The EUMETSAT use case also proposes processes and workflows that should be supported by the GDC API, catering for common processing steps that most users need to apply to the data. Interactions with external systems such as other data cubes and web mapping services are also foreseen.

- ECMWF's use case considered the interoperability between the Meteorological Archival and Retrieval System (MARS) and a potential candidate for the implementation of a GDC API. This use case covers requirements with data cubes of arbitrary dimension, of different types, and irregular or sparse data content. ECMWF also requested support for non homogeneous grids and non-orthogonal data access.

Please note that Testbed 19 is not the first Testbed with a research thread focused on GDCs. Previous efforts and especially the resulting ERs give some clues into gaps that need to be addressed.

- The OGC Testbed-17: Geo Data Cube API Engineering Report (ER) (21-027) is, like all OGC Collaborative Solutions and Innovation Program ERs, available from the OGC public ER webpage. The OGC Testbed-17 ER documents the results and recommendations of the Geo Data Cube API task. The Testbed-17 Call for Participation provides additional insights into requirements and use cases. The ER defines a draft specification for an interoperable GDC API leveraging OGC API building blocks and details implementation of the draft GDC API. The ER also explores various aspects including data retrieval and discovery, cloud computing, and Machine Learning. Implementations of the draft GDC API were demonstrated with use cases including the integration of terrestrial and marine elevation data and forestry information for Canadian wetlands. One of the key requirements for future work highlighted was as follows.
 - “Defining well-known processes expecting specific inputs including a particular convenient processing language to facilitate flexible coverage processing should be considered.”
- OGC Testbed-16: Data Access and Processing ER (20-016) summarizes the results of the 2020 Testbed-16 Data Access and Processing task. The task had the primary goal to develop methods and apparatus to simplify access to, processing of, and exchange of environmental and Earth Observation (EO) data from an end-user perspective.
 - “Consider adding the ability to negotiate output formats.”
 - “Explore adding support for HTTP POST for invoking DATA processes.”
 - “Considering adding an up/downsampling capability of data to control the volume of data being processed.”
 - “Investigate DAPA extensibility.”
 - “Enhanced and interactive documentation about all aspects of a DAPA deployment.”

- “The ability to combine different datasets in one request.”

Additionally, there are several key developments happening in parallel both inside and outside of the OGC. Within the OGC, there are the activities focused on the various components of the OGC API suite of Standards, including common, coverages, features, records, processing and, of course, the original OGC Web Services (W*S) suite of standards. Outside the work of the OGC, there are specifications such as STAC and openEO tackling similar objectives. Especially STAC, but also openEO, has already seen a considerable uptake in industry and space agencies supported by a number of open-source implementations available for use. To this end, when starting to define solutions and draft a GDC API standard, all of these technologies and existing GDC related specifications should be considered. In the following section on the state of the art, a more detailed comparison of core parts of the existing specifications is presented in the form of crosswalks.

3

STATE OF THE ART

This chapter provides an overview of relevant technologies and specifications of useful GeoDataCube API's evaluated in Testbed-19. Crosswalks are used to document detailed comparisons between key specifications/standards such as openEO, STAC, part of the OGC API suite, and WCPS. These crosswalks highlight architectural, conceptual, and requirements differences and try to map the most relevant pieces to each other. These crosswalks were created prior to writing the first draft version of the GDC API specification. This approach was taken to better understand the landscape of existing technologies, and all testbed participants have been invited to develop relevant comparisons. As a result, the following crosswalks were created.

3.1. Crosswalk between STAC and OGC API – Records

This crosswalk provides a brief overview over similarities and differences between the following.

- **STAC (API), v1.0.0** (+ extensions)
- **OGC API – Records – Part 1: Core, draft version (2023-05-26)**

In the following, **OAR1** is used as an abbreviation for **OGC API – Records – Part 1**.

3.1.1. Introduction

This crosswalk compares the content model (STAC vs. OAR1) and the API specification (STAC API vs. OAR1).

Unfortunately, STAC and OGC API – Records are not fully compliant with each other. This means that by default a STAC API endpoint does not automatically conform to a OGC API – Records endpoint and vice versa. Nevertheless, there are no major conflicts in the specifications so that both implementations can interoperate with a relatively small number of additions.

3.1.1.1. Terminology

The different entities specified in STAC and OAR1 can be mapped as follows.

Table 1

STAC	OAR1
Catalog	Record Collection
Collection	Record Collection
Item	Record


Nevertheless, there may be other mappings between concepts and properties in the specifications. Some may argue a STAC Collection is a OAR1 Record. Due to the way the JSON representations are defined, a mapping as in the table above is assumed.

Please also note that Catalogs, Collections, and Items could be Records in OAR1 and some implementations only convert Records to Record Collections once there is at least one child record. Catalogs are currently mostly used in static deployments and do not play a huge role in APIs/dynamic deployments.

3.1.2. Static (content)

- The content model and static catalog behavior for STAC is specified in the standalone STAC specification and corresponding extensions.
- The content model and static catalog behavior for OAR1 is specified in the requirement classes “Record Core,” “Record Collection,” and “Crawlable Catalog.”

Note: In the tables below, the following applies.

- **bold** properties indicate that the properties are required.
- ***bold and italic*** properties are required if a criteria is met
- means that fields are fully compatible.
-  means that fields are generally compatible (if applicable) but the field is not required in both specs or there are other minor differences implementors need to be aware of.
- **✘** means there is a conflict between the specifications: the generated JSON is NOT compatible.
- **Profile*** Checks the compliance if STAC would be defined as an OGC API – Records – Part 1 profile / extension.

3.1.2.1. Catalogs

Assuming a JSON (non-GeoJSON) encoding.

Table 2

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
type = Catalog	type = Collection	⚠ Issue	⚠ There is no Catalog in OAR1, a “lightweight” Collection may be used instead.
stac_ version	-	✓	⚠
stac_ extensions	-	✓	✓
id	id	✓	✓
title	title	⚠ Issue	⚠
description	description	✓	⚠ STAC disallows empty strings
links	links	⚠ Issue 1 Issue 2 Issue 3	⚠ Structure is compatible except for templated links and different relation type requirements (none in STAC; self in OAR1). Also, OAR1 and STAC both use the relation type child with a media type application/json to link to catalogs, but clients cannot distinguish whether they can expect OAR1 or STAC.

See below for more details about Collections.

3.1.2.2. Collections

Assuming a JSON (non-GeoJSON) encoding.

Table 3

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
type = Collection	type = Collection	✓	✓
-	itemType = record (API only?)	✓	✓

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
stac_version	-	✓	⚠
stac_extensions	conformsTo (tbc)	✓	⚠ STAC extensions could be used in conformsTo, but conformance classes that are not valid JSON schemas can't be used in stac_extensions.
id	id	✓	✓
title	title	⚠ Issue	⚠
description	description	✓	⚠ STAC disallows empty strings
links	links	⚠ Issue 1 Issue 2 Issue 3	⚠ Structure is compatible except for templated links, but different relation type requirements (none in STAC; self and root in OAR1). Also, OAR1 and STAC both use the relation type child with a media type application/json to link to collections, but clients cannot distinguish whether OAR1 or STAC is expected.
keywords	keywords	✓	✓
license	license	✓ (STAC 1.1+)	⚠ STAC 1.0 is not fully compatible, but STAC 1.1 will be made compatible.
providers	-	✓	✓
contacts (extension)	contacts	✓	✓
extent	extent	✓	⚠ STAC requires a bounding box and a temporal interval.
summaries	-	✓	✓
assets	-	✓	⚠ Assets are provided as links (aka "associations") in OAR1.
language (extension)	language	✓	✓
languages (extension)	languages	✓	✓
-	recordLanguages (API only?)	✓	✓
created (common metadata)	created	✓	✓

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
updated (common metadata)	updated	✓	✓
themes (extension)	themes	✓	✓
crs (API only)	crs	✓	✓
-	rights	✓	✓

3.1.2.3. Items / Records

The following assumes a GeoJSON encoding for Records.

3.1.3. Top-level

Table 4

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
type = Feature	type = Feature	✓	✓
stac_version	-	✓	⚠
stac_extensions	conformsTo	✓	⚠ STAC extensions could be used in conformsTo, but conformance classes that are not valid JSON schemas cannot be used in stac_extensions.
id	id	✓	✓
geometry	geometry	✓	⚠ STAC <u>disallows GeometryCollections</u>
bbox	bbox	✓	⚠ STAC requires bbox if geometry is not null.
properties	properties	✓	
links	links	⚠ Issue 1 Issue 2 Issue 3	⚠ Structure is compatible except for templated links, but different relation type requirements (none in STAC; self in OAR1). Also, OAR1 and STAC both use the relation type item with a media type application/geo+json to link to items/records, but clients cannot distinguish whether OAR1 or STAC is expected.

STAC	OAR1	PROFILE*	COMPATIBLE / COMMENTS
assets	-	✓	⚠ Assets are provided as links (aka “associations”) in OAR1.
collection	-	✓	✓ This field is <i>required</i> in STAC if such a relation type is present and is <i>not allowed</i> otherwise.
-	time	⚠ Issue 1 Issue 2	⚠ STAC: datetime / start_datetime / end_datetime in properties. STAC can’t encode all options that OAR1 allows.

3.1.4. Properties

Table 5

STAC	OAR1	PROFILE*	COMPATIBLE
-	type	⚠ Issue	⚠
datetime / start_datetime / end_datetime	-	⚠ Issue 1 Issue 2	⚠ OAR1: time in the top-level object
title (common metadata)	title	⚠ Issue	⚠
description (common metadata)	description	✓	⚠ STAC disallows empty strings
keywords (common metadata)	keywords	✓	✓
license (common metadata)	license	✓ (STAC 1.1+)	⚠ STAC 1.0 is not fully compatible, but STAC 1.1 will be made compatible.
created (common metadata)	created	✓	✓
updated (common metadata)	updated	✓	✓
contacts (extension)	contacts	✓	✓
themes (extension)	themes	✓	✓
language (extension)	language	✓	✓
languages (extension)	languages	✓	✓
-	resource Languages	✓	✓
-	externalIds	✓	✓

STAC	OAR1	PROFILE*	COMPATIBLE
-	rights	✓	✓
-	formats	✓	✓ STAC: A similar list can be obtained from the type field in assets

3.1.5. API (behavioral)

- The STAC API description is specified in the standalone STAC API specification and corresponding extensions (conformance classes Core, Collections, Features).
- The OAR1 API description is specified in the requirement classes “Records API,” “Searchable Catalog,” and “Local Resources Catalog.”

Both API descriptions have dependencies on the OGC API – Features and OGC API – Common Standards.

Both API specifications use HTTP as the basis and encourage the use of HTTPS. HTTP 1.1 is required for OAR1, while the STAC API description does not explicitly define an HTTP version. Both API specifications follow REST principles and make use of HTTP content negotiation. Both APIs make broad use of “Web Linking” (compatible between OAR1 and STAC API). Both specifications recommend the implementation of CORS.

The default encoding for request and response bodies is JSON. The OGC API – Common Standard recommends also supporting HTML as an response body encoding. Therefore content negotiation needs to be implemented more carefully for OAR1. A STAC API implementation usually uses client software to render HTML output from JSON (e.g., STAC Browser).

Both specifications make broad use of OpenAPI 3.0 (or later) and JSON Schema for specification purposes.

3.1.5.1. Landing Page

- STAC: GET / (required)
- OAR1: GET / (required)

As both the STAC and OAR1 landing pages are consistent with the requirements specified in the [OGC API – Common Standard](#), the landing pages are very similar.

In OAR1 an implementation must provide just links and can optionally add title and description.

An implementation of the STAC API requires additional properties (`stac_version`, `type`, `id`, `description`) to form a full STAC Catalog. The STAC API implementation also lists the conformance test classes in the landing page, while OAR1 has the implementation separate.

The use of links is a bit different in STAC API and OAR1 implementations. The implementation of the Collection List is optional in STAC. A STAC API implementation can also just expose child links to catalogs/collections and/or to a search endpoint.

3.1.5.2. Conformance

- STAC: GET `/conformance` (optional)
- OAR1: GET `/conformance` (required)

Both endpoints are 100% equivalent. OAR1 requires a separate endpoint that lists conformance test classes, which is optional in STAC API. A STAC API implementation additionally lists the conformance classes in the landing page.

3.1.5.3. Collection List

- STAC: GET `/collections` (optional)
- OAR1: GET `/collections` (required)

As the endpoints for collection lists are both based of [OGC API – Features – Part 1](#), the endpoints are very similar. The difference between STAC and OAR1 is how the individual collections are encoded, see Collections for details.

3.1.5.4. Individual Collection

- STAC: GET `/collections/{collectionId}` (optional)
- OAR1: GET `/collections/{collectionId}` (required)

As the endpoints for individual collections are both based of [OGC API – Features – Part 1](#), the endpoints are very similar. The difference between STAC and OAR1 is how the individual collections are encoded, see Collections for details.

3.1.5.5. Item List

- STAC: GET `/collections/{collectionId}/items` (optional)
- OAR1: GET `/collections/{collectionId}/items` (required)

As the endpoints for item lists (per collection) are both based of [OGC API – Features – Part 1](#), the endpoints are very similar. The difference between STAC and OAR1 is how the individual items are encoded, see [Items](#) for details. Additionally, OAR1 defines the following parameters that implementations must support: `q`, `type`, and `externalId`.

3.1.5.6. Individual Item

- STAC: GET `/collections/{collectionId}/items/{itemId}` (optional)
- OAR1: GET `/collections/{collectionId}/items/{recordId}` (required)

As the endpoints for individual items are both based on top of [OGC API – Features – Part 1](#), the endpoints are very similar. The difference between STAC and OAR1 is how the individual items are encoded, see [Items](#) for details.

3.1.5.7. Search

STAC defines three ways to search for resources:

- **Collection Search Extension:** Search for collections via GET `/collections` (based on Local Resource Catalogue in OAR1);
- **Item Search:** Search (globally) for items across collections via GET `/search` and POST `/search`; and
- **Items per Collection:** Filter for items in a specific collection via GET `/collections/{collectionId}/items` (see [Item List](#)).

OAR1 defines similar ways to search for resources:

- **Local Resource Catalogue:** Search for record collections via GET `/collections`;
- **Record Search:** A global search for records is planned in the requirement class [Searchable Catalogue](#), but is not yet fully defined. There are [chances for incompatibilities with STAC](#); and
- **Records per Record Collection:** Filter for records in a specific record collection via GET `/collections/{collectionId}/items` (see [Item List](#)).

3.1.5.8. Datacubes

OAR1 does not prescribe how to provide metadata about datacubes. OGC in general has a couple of related standards, e.g., [OGC API Coverages](#) and the draft [GeoDataCube API](#), but the relation with OAR1 is not clearly defined for any of them.

STAC does not provide metadata about datacubes either in the core, but it has a [datacube extension](#).

3.2. Crosswalk between the openEO API specification and OGC API – Processes Standard

The crosswalk provides a brief overview of the similarities and differences between the following.

- **openEO API, v1.2.0**
- **OGC API – Processes – Part 1: Core, v1.0.0**

In the following, **OAP1** is used as an abbreviation for **OGC API – Processes – Part 1: Core Standard**.

3.2.1. Introduction

The OGC API – Processes – Part 1: Core Standard (aka Processes API or OAPI) only defines processing, while the openEO API has a much broader scope. openEO covers many parts that other OGC API Standards define: some are aligned and some are not.

Conceptually, the two API specifications are similar but have some conflicts that cannot be easily resolved (e.g., process description with multiple outputs in OAP1, job listing with different job status values).

A key differentiator between OAP1 and openEO is that process chaining is a fundamental concept in openEO for defining workflows while OAP1 is more meant to run larger “black box” workflows. However, Part 3 of OGC API – Processes defines the behavior of an implementation that supports the ability to define ad-hoc workflows, chains nested processes, refers to both local and external processes and collections of data accessible via OGC API standards as inputs to a process, and triggers execution of processes through OGC API data delivery specifications such as OGC API – Tiles, DGGs, Coverages, Features, EDR, and Maps. This extension also defines the behavior of an implementation that supports the ability to deploy a workflow defined using the mechanisms it describes through the OGC API – Processes – Part 2: Deploy, Replace, Undeploy extension.

Another key differentiator is that openEO has a list of [pre-defined but extensible processes](#) available while the OGC API – Processes Standard does not predefine processes.

As such, the target audience for implementing and using OAP1 and openEO may only partially overlap.

The openEO API specification covers the following “categories” of endpoints.

- API discovery – Partially covered by OGC API – Processes – Part 1
- Authentication – Not defined by the OGC API Standards Suite
- Data Discovery – Covered by various other OGC API Standards (Coverages, EDR, Features, Records, ...)
- Process Discovery – Covered by OGC API – Processes – Part 1 Standard
 - Pre-defined processes – Covered by OGC API – Processes – Part 1 Standard
 - User-defined processes / Workflows – Covered by draft OGC API – Processes – Part 2 and 3 extensions.
- Data Processing – Covered by draft OGC API – Processes – Part 1 and 3 extensions.
 - Synchronous processing – Covered by the OGC API – Processes – Part 1 Standard
 - Batch Job processing – Covered by OGC API – Processes – Part 1 Standard
 - On-demand processing – Covered by other OGC API Standards (Maps, Tiles, ...)
- File Storage – Not covered by the OGC API Standards Suite

3.2.2. General API mechanics

Both API definitions use HTTP as the basis and encourage the use of HTTPS. HTTP 1.1 is required for an OAP1 implementation, while openEO does not specify a specific HTTP version. Both API specifications follow REST principles and make use of HTTP content negotiation. Both API specifications make broad use of “Web Linking” (compatible between OAP1 and openEO). Both specifications recommend the implementation of Cross-Origin Resource Sharing (CORS).

The default encoding for request and response bodies is JSON. The OGC Processes API recommends supporting HTML as a response body encoding. openEO uses client software to render HTML output from JSON.

Both specifications make broad use of OpenAPI 3.0 and JSON Schema for specification purposes.

Many OAP1 and openEO API endpoints support pagination through links with the relation types `next` and `prev`. These endpoints have a `limit` parameter to enable pagination for a specific page size. **Note:** Pagination is rarely used in openEO implementations and most clients don't support it consistently.

3.2.3. API discovery

Discovering an API endpoint, connecting to it and reading the capabilities is always the first step clients need to execute.

3.2.3.1. Well-known document

- openEO: GET /.well-known/openeo
- OAP1: n/a

openEO clients usually first connect to the well-known document to discover different versions or implementations of a server. Thus a client can choose a suitable API version including choosing between production and development instances. Then the openEO client connects to the selected instance's landing page. OGC API – Processes clients always directly connect to a landing page. openEO clients can also directly connect to the landing page.

This folder structure with the document typically resides outside of the actual API implementations, i.e., in the root of the host or domain.

3.2.3.2. Landing page

- openEO: GET / (required)
- OAP1: GET / (required)

As the landing pages are both consistent with OGC API – Common, the landing pages are very similar.

Some differences include the following.

- openEO requires the following additional fields.
 - Defined in OAP1, but not required: title, description
 - Not defined in OAP1: api_version, backend_version, stac_version, id, endpoints, (type)
- The relation type to link to the conformance classes is conformance in openEO (originating from OGC API – Features / STAC API) and <http://www.opengis.net/def/rel/ogc/1.0/conformance> in OAP1. Two links with the same target but different relation types will be required.
- The existence of API endpoints in openEO is primarily checked through endpoints, which is not present in OAP1. OAP1 uses links, which openEO primarily uses for non-API-related links. Nevertheless, links such as required in OAP1 can be added easily to an openEO implementation. The following additional links are relevant for OAP1.
 - One of the link relations service-desc or service-doc is required in OAP1 and should link to an API description (e.g., OpenAPI or rendered HTML version).

- A link with relation type <http://www.opengis.net/def/rel/ogc/1.0/processes> to /processes is required for OAP1, but not in openEO.
- A link with relation type <http://www.opengis.net/def/rel/ogc/1.0/job-list> to /jobs can be added in OAP1, but is not present in openEO.

3.2.3.3. Conformance classes

- openEO: GET /conformance (optional)
- OAP1: GET /conformance (required)

Both conformance test class endpoints are 100% equivalent. OAP1 requires a separate endpoint that lists conformance test classes. openEO additionally supports listing the conformance test classes in the landing page (follows STAC API). Conformance test classes are only fully defined since openEO API v1.2.0.

3.2.4. Authentication

- openEO: GET /credentials/basic and/or GET /credentials/oidc
- OAP1: n/a

OpenEO defines two authentication mechanisms:

- OpenID Connect (primary); and
- HTTP Basic (secondary, primarily for development/testing purposes).

The OAP1 Standard does not define any authentication mechanisms, but both authentication mechanisms can also be implemented for OAP1. The main issue will be that OAP1 clients will likely not support the mechanisms.

The availability of the authentication mechanisms is detected through the endpoints in the openEO landing page, while in OAP1 the OpenAPI document for such an authentication mechanism (which implicitly requires a link to an OpenAPI 3.0 file with relation type service-desc in the landing page) must be parsed.

3.2.5. Data Discovery

Data Discovery is not covered by the OAP1 Standard, but can be “plugged in” by implementing various other OGC API building blocks (e.g., Coverages, EDR, Features, Records, etc.). Except for OGC API – Processes – Part 3, it is not clear how the processes defined in the OAP1 Standard can access resources from other API implementations. The processes probably need to implement data access individually through clients implementing OGC API building blocks.

As such, OAP1 could also be made compliant with STAC API – Collection and STAC API – Features. STAC API – Collections is required by openEO. STAC API – Features is optional in openEO, but would likely be required to allow data access through an OAP1 processes.

3.2.6. Process Discovery

The OAP1 Standard claims that it

does not mandate the use of any specific process description to specify the interface of a process.

The OAP1 Standard does define a “default” encoding in the conformance class “OGC Process Description.” Unfortunately, this is a somewhat misleading statement and the OAP1 Standard still provides a predefined set of fields which conflict with openEO (see below).

The openEO API specification defines a single encoding for process descriptions.

An important difference is that in OAP1 a single process is executed (optionally processes can be chained using the draft OGC API – Processes – Part 3 extension) which means a process in an OAP1 implementation is often more complex. In an openEO implementation a process is often very fine-granular (e.g., addition of two numbers) and can usually chain multiple of processes to a more complex process (graph). The process chaining into a full graph is fundamental in openEO.

Another major difference is that in openEO there are pre-defined process descriptions available for common use cases. The OGC API – Processes Standard does not provide pre-defined process descriptions. In theory, an implementation could re-use the openEO processes in an OGC API – Processes endpoint if OGC API – Processes – Part 3 is implemented for chaining.

3.2.6.1. Pre-defined processes

- openEO: GET /processes (required)
- OAP1: GET /processes, GET /processes/{processID} (both required)

In openEO GET /processes operation returns the full process metadata, while an OAP1 implementation only returns a summary. The general structure of the response of GET /processes is the same (processes and links). The OGC Process Description and the openEO process description are not compatible.

openEO does not yet define an endpoint to retrieve an individual process such as a OAP1 GET /processes/{processID} endpoint will. There is a proposal available to add such an endpoint to the openEO specification. openEO has the concept of namespace for processes though and thus defines the endpoint at GET /processes/{namespace}/{process} which would conflict with the OAP1 definition.

Some notable differences in the process description (only fully delivered via GET /processes/{processID}) include the following.

- OAP1 defines `jobControlOptions`, which is undefined in openEO yet (which implies ["sync-execute", "async-execute"]).
- OAP1 defines `outputTransmission`, which is not available in openEO. It seems as though this will be removed.
- OAP1 allows multiple outputs, openEO only allows a single output (potential workaround: wrap in an array or object).
- OAP1 uses `title` and openEO uses `summary`.
- OAP1 specifies inputs as object and the identifier as key, openEO specifies parameters as array and the identifier as name property (but the content of each is otherwise very similar).

Below is a simple process encoded in the two variants discussed here.

- [OGC Process Description](#)
- [openEO Process](#)

3.2.6.2. User-defined processes / Workflows

Workflows are not described in the OGC API – Processes – Part 1 Standard. Instead, the draft Part 2 extension defines the deployment/management of workflows and the Part 3 extension specifies how to define the workflows. The Part 2 extension defines how to chain/nest processes in an OGC API – Processes implementation and also defines a “workflow language” (“Modular OGC API Workflow JSON”, short: MOAW), but it seems to also allow providing other workflow languages such as openEO user-defined processes and the Common Workflow Language (CWL).

openEO defines the `/process_graphs` endpoints and has workflows (called “user-defined processes” in openEO, i.e., a process with a process graph) included in the core.

Related documents:

- MOAW: https://github.com/opengeospatial/ogcapi-processes/blob/master/extensions/workflows/sections/clause_6_overview.adoc (and following chapters)
- openEO: https://github.com/opengeospatial/ogcapi-processes/blob/master/extensions/workflows/sections/clause_13_openeo_workflows.adoc
- CWL: https://github.com/opengeospatial/ogcapi-processes/blob/master/extensions/workflows/sections/clause_12_cwl_workflows.adoc

A comparison between MOAW and openEO user-defined processes should also be considered in the future.

3.2.7. Data Processing

3.2.7.1. Synchronous processing

- openEO: POST /result
- OAP1: POST /processes/{processID}/execution

3.2.7.2. Batch Job processing

3.2.7.2.1. Job list

- openEO: GET /jobs
- OAP1: GET /jobs

The job list provided by an OAP1 endpoint has several parameters that are not available in an openEO implementation. The general structure of the response of GET /jobs is the same (jobs and links). Each job complies to the structure discussed in Job status.

3.2.7.2.2. Job status

- openEO: GET /jobs/{jobID}
- OAP1: GET /jobs/{jobID}

Similar properties in OAP1 and openEO:

- jobID (required) / id (required)
- processID (string) / process (required, object containing chained processes)
- status (required, below: OAP1 / openEO) – The values map as follows:
 - accepted / created
 - n/a / queued
 - running / running
 - successful / finished
 - failed / error

- dismissed / canceled
- created (required in openEO only)
- updated
- progress
- links (to be added to openEO)

Additional properties in the OAP1 Standard:

- type (required) (always: process)
- message (-> use logs in openEO)
- started (-> use logs in openEO)
- finished (-> use logs in openEO)

Additional properties in the openEO specification:

- title
- description
- costs
- budget
- usage

3.2.7.2.3. Creating a job

- openEO: POST /jobs
- OAP1: POST /processes/{processID}/execution

An OAP1 implementation may provide inputs, outputs, the response type (raw or document), and/or a subscriber.

An openEO implementation must provide a process (chained processes as process graph with optional additional metadata) and may additionally provide a title, a description, a billing plan, and/or a maximum budget.

Both specifications seem to support providing additional properties.

3.2.7.2.4. Queueing a job

- openEO: POST /jobs/{jobID}/results
- OAP1: n/a (queued directly after creation)

3.2.7.2.5. Cancel / Delete a job

- openEO: DELETE /jobs/{jobID}/results / DELETE /jobs/{jobID}
- OAP1: n/a / DELETE /jobs/{jobID}

3.2.7.2.6. Result Access

- openEO: GET /jobs/{jobID}/results
- OAP1: GET /jobs/{jobID}/results

In an openEO implementation, result access fully relies on STAC. The response is a valid STAC Item or Collection.

For an OAP1 implementation the result access differs depending on the given parameters.

3.2.7.3. On-demand processing

On-demand processing in this context means that processing is only executed for extents shown to the user, e.g., on a web map. Results are processed “on demand” depending on the interaction with the map (e.g., how Google Earth Engine does it in their Code Editor by default).

TBD, lower priority as it's not covered by OAP1. It's also optional and very different in openEO. Instead it is defined as a separate OAP1 API implementation for individual OGC API resources. For example, a combination of OGC API – Maps and OGC API – Processes.

3.2.8. File Storage

- openEO: /files (various methods), /files/{path} (various methods)
- OAP1: n/a

As file storage is not specified in the OAP1 Standard or in any of the OGC APIs, comparisons cannot be provided.

3.3. Crosswalk between the openEO API specification and the draft OGC API – Coverages Standard

The following crosswalk provides a brief overview of the similarities and differences between:

- [openEO API, v1.2.0](#)
- [OGC API – Coverages – Part 1: Core, v0.0.6](#)

3.3.1. Introduction

The draft OGC API – *Coverages* Standard (OAC1 or Coverages API) establishes how to access coverages as defined by the [OGC Abstract Specification Topic 6 / ISO 19123 Schema for coverage geometry and functions](#), and thus provides an alternative to the established OGC [Web Coverage Service \(WCS\)](#) interface standard (v2.1).

The strategy behind the development of the OGC API – Coverages Standard is to define a minimal, relatively simple Coverages API Standard first, with new capabilities being incrementally added based on community demands while defining how to access (portions of) coverages from a catalog. The content of the coverage can be encoded using any suitable logical model and physical encodings, such as those defined by [Coverage Implementation Schema \(CIS\) 1.1](#), [netCDF](#), [CoverageJSON](#), [GeoTIFF](#) (including [Cloud Optimized GeoTIFF \(COG\)](#)), or [LAS/LAZ](#) (including [Cloud Optimized Point Cloud \(COPC\)](#)).

The following crosswalk compares the Draft Coverages API Standard with the openEO API, touching all the functionalities that are declared in the OGC standard.

- (meta)data retrieval
- subsetting through dimension(s)
- range (bands) subsetting
- up/down-scaling
- data retrieval as tiles

The following abbreviations will be used throughout this chapter.

- **openEO** : *openEO API, v1.2.0*
- **OAC1** : *OGC API – Coverages – Part 1*

3.3.2. Nomenclature

Despite both specifications having dependencies on the [OGC API – Features Standards](#), openEO/STAC and OGC have different terminologies and data models. Therefore, below are definitions of the most relevant terms from both API “worlds.”

Most notably, an OGC *coverage* – even though formally defined as a *feature* (and not a *collection* thereof) – is frequently an abstraction for a *collection* of (generally, but not limited to, gridded) datasets/items (like time-series of satellite images for instance), so the concept “coverages” can generally be compared with STAC *collections*, especially *datacubes*.

3.3.2.1. OGC

collection:

A set of **features** from a dataset.

coverage:

A **feature** that acts as a function to return values from its **range** for any direct position within its spatial, temporal, or spatio-temporal **domain**; this can be equivalent to either a [STAC Item](#) or [Collection](#).

domain:

A set of direct positions along some *axis* in space and time (or other dimensions) that defines the “location” of a coverage, associated with a (compounding of) spatio(-temporal)(-other) coordinates reference system (CRS), and for each of which a set of 1+ values (**range**) are associated.

feature:

An abstraction of real world phenomena.

range:

A set of **feature** attribute values associated by a function with the elements of the **domain** of a coverage, that means the stored values of a coverage like for instance the spectral bands; in the STAC [datacube](#) extension, **range** and **domain** are condensed into the same **cube:dimensions** container.

3.3.2.2. STAC

catalog:

A logical group of other **catalog**, **collection**, and **item** objects.

collection:

A set of common fields to describe a group of **items** that share properties and metadata

item:

A GeoJSON Feature augmented with foreign members relevant to a STAC object.

cube:dimensions:

A set of dimensions that can be of different types: spatial, temporal, “other,” etc.

3.3.3. Crosswalk

The capabilities defined in the **openEO** specification and the draft **OAC1** Standard largely overlap. However, as also declared in the preface, OAC1 defines a simpler service as specified in the OAC1 core requirements class (no authentication/authorization management, processes, batch jobs, logs, processing budgets, etc.).

The **draft OAC1** Standard usually specifies simpler ways to request basic operations on datasets (like filtering or resampling) which is done by using the query parameters of the URLs. On the other hand, **openEO** permits a much powerful service with processing graphs and asynchronous jobs management, but at the cost of a slightly more laborious way to execute the most basic “processing” operations.

The coverage data model is generally a bit more flexible with regards to the “arbitrary” non-spatial non-temporal dimensions, while the processes in **openEO** generally do not include such dimensions.

The only **OAC1** capability that **openEO** does not directly provide (if not via cascade with a secondary tiles service) is to attach tilesets to coverages, and hence enable tiled access to the data.

3.3.3.1. Meta / API capabilities

In this section the available information on the API capabilities is provided. This information is relevant for interoperability and “machine-actionable” services.

3.3.3.2. Well-known URIs

- openEO: [GET /.well-known/openeo](#) (optional)
- OAC1: n/a

Site-wide metadata information: In an **openEO** API implementation this is a list of versions of the API that the server supports, and it is meant to help a client choose the most suitable endpoint (eg. production/development).

Site-wide metadata is not specified in the draft **OAC1** Standard.

3.3.3.3. Landing page

- openEO: `GET /` (required)
- OAC1: `GET /` (required)

The landing pages as defined in both specifications are consistent with the [OGC API – Common Standard](#), and there are only minor differences in the returned schemas.

3.3.3.4. Conformance

- openEO: `GET /conformance` (optional) (only from v1.2.0)
- OAC1: `GET /conformance` (required)

Both endpoints are 100% equivalent (**openEO** additionally supports the ability to list the conformance test classes in the landing page, following the STAC API).

3.3.3.5. API description

- openEO: as per `service-desc/service-doc` relation types
- OAC1: `GET /api` (required)

Both **openEO** and **OAC1** implementations can provide service descriptions (e.g., OpenAPI documents) and human-readable documentation in the resources pointed to by the standard `service-desc` and `service-doc` relation types. In an **OAC1** implementation such resources (both) point to the `/api` endpoint.

3.3.3.6. Authentication

- openEO: `GET /credentials/basic` and/or `GET /credentials/oidc`
- OAP1: n/a

The **openEO** specification defines two authentication mechanisms: [OpenID Connect](#) (primary) and HTTP Basic (for development/testing). The draft **OAC1** Standard does not define any authentication mechanisms. While both mechanisms could be implemented, **OAC1** clients will likely not support them.

The availability of the authentication mechanisms is detected through the endpoints in the **openEO** landing page, while in **OAC1** parse the OpenAPI document needs to be parsed for the

authentication information (which implicitly requires a link to an OpenAPI 3.0 file with relation type `service-desc` in the landing page).

3.3.3.7. Data discovery

- openEO:
 - [GET /collections](#) (required)
 - [GET/POST /search](#) (optional)
- OAC1:
 - [GET /collections](#) (required)

openEO and **OAC1** implementations both provide a resource for accessing the whole catalog of datacubes/coverages offered by the server. This includes basic/abbreviated metadata for each of the assets in order to try to keep the response size small. Full metadata details are instead available as per data description.

CIS coverages will be returned by an **OAC1** request, STAC Collections by an **openEO** request.

Optionally, **openEO** specifies how to search for individual STAC items “hidden” inside the collections, as per [STAC API repository](#).

3.3.3.8. Data description

- openEO:
 - [GET /collections/{collectionId}](#) (required)
 - [GET /collections/{collectionId}/queryables](#) (optional)
- OAC1:
 - [GET /collections/{collectionId}](#) (required)
 - [GET /collections/{collectionId}/coverage/rangeType](#) (required)
 - [GET /collections/{collectionId}/coverage/domainSet](#) (required)
 - [GET /collections/{collectionId}/coverage/metadata](#) (optional)

Full metadata descriptions of collections are available in both the **openEO** and **OAC1** specifications, encoded as STAC Collections or CIS Coverages, respectively.

Additional links are provided (required) for coverages in an **OAC1** implementation with respect to the basic “Collection Resources” as defined in the OGC API – Common Standard: One for the description of the sole “range” of the coverage, and one for the sole “domain,” which is just a subset of the full metadata description. Also available, though optional, is the link to the sole coverage’s “metadata,” meaning supplementary/domain-specific extra metadata fields that might be attached to a coverage.

The **openEO** specification offers a more flexible and machine-actionable solution for filtering the description of a collection. This is by means of the “queryables”: a listing of all metadata filters available for each collection. This is in line with both [filter](#) STAC API extension and [the draft OGC API – Features – Part 3: Filtering Standard](#).

3.3.3.9. Data access

- openEO:
 - [POST /result @ load_collection](#) process (sync)
 - [POST /jobs @ load_collection](#) process
[POST /jobs/{jobId}/results](#) (async)
- OAC1:
 - [GET /collections/{collectionId}/coverage](#) (required)
 - [GET /collections/{collectionId}/coverage/rangeset](#) (optional)

OAC1 : Encoding as negotiated in the HTTP protocol, with a proper fallback media type otherwise. The actual coverage values (the range set) can be represented either in-line or by reference to an external file which may have any suitable encoding.

An **openEO** implementation provides access to the actual data by means of the [load_collection](#) process, which can be submitted either synchronously or asynchronously as a batch job. The format of the resulting data shall always be specified by chaining a [save_result](#) process.

For pre-filtering/data subsetting options, see next section.

3.3.3.10. Data subsetting

- openEO:
 - [POST /result @ load_collection](#) process (sync)
 - [POST /jobs @ load_collection](#) process
[POST /jobs/{jobId}/results](#) (async)
- OAC1:

- `bbox / datetime / subset` query params (domain subsetting)
- `properties` query param (range subsetting)

The parameters offered by the **openEO** `load_collection` process supports subsetting the requested collection along a cube's dimensions, which also includes what the **OAC1** Standard calls the range set, hence the "bands" of the dataset. There is no option for low:high subsetting of an arbitrary dimension which might neither be classified as spatial nor temporal, like the `subset` query parameter offers in **OAC1**.

Spatial subsetting as defined in the **openEO** specification is more powerful as not only bounding-boxes but also GeoJSON geometries can be specified for a more accurate clipping of the data. On the other hand, **OAC1** supports specifying open/unbounded spatial intervals. Open/unbounded time intervals are supported in both API specifications (via `null` or double-dots `..`, respectively).

Spatial filtering with coordinates specified with an arbitrary CRS is available in both API specifications. In both cases, all data values that spatially *intersect* with the provided spatial extent/range (boundaries included) will be in the response.

Subsetting of an arbitrary dimension – not classified as spatial not temporal, nor band in the openEO datacube model – is not possible in an **openEO** implementation, while available in an **OAC1** implementation via `subset` parameters.

In case the response is too large, the current proposal is for an **OAC1** implementation to return a URI instead of the content, or a reduced-resolution coverage with a URI to the full resolution response.

Filtering/cherry-picking of bands is available in both the **OAC1** and **openEO** implementations via the `properties` parameter and `load_collection` process, respectively. Both band names and band indices can be used in an **OAC1** implementation, while an **openEO** implementation only accepts band names.

An **openEO** compliant server can optionally provide a rough estimate of the time/cost of a stored job via `/jobs/{jobId}/estimate`. Server logs are also available from an **openEO** server implementation either directly in the synchronous processing response (as a link), or via the API resources dedicated to the batch jobs (eg. `jobs/{jobId}/logs`).

3.3.3.11. Data scaling

- openEO:
 - `POST /result @ resample_spatial` process (sync)
 - `POST /jobs @ resample_spatial` process
`POST /jobs/{jobId}/results` (async)
- OAC1:

- GET /collections/{collectionId}/coverage?scale-size=axisName({number})[,axisName({number})]*
- GET /collections/{collectionId}/coverage?scale-axes=axisName({number})[,axisName({number})]*
- GET /collections/{collectionId}/coverage?scale-factor={number}

Specifying how to rescaling a dataset in an **openEO** implementation requires the same procedure as for any other kind of processing, thus preparing a process graph that executes the desired operation, then fetch the results, either synchronously or asynchronously through the creation of a [batch job](#).

At of February 2024, the available **openEO** specified processes for data resampling are [resample_spatial](#) – that rescales the spatial dimensions to a target resolution, or to a target spatial coordinate reference system – and [resample_cube_spatial](#)/[resample_cube_temporal](#), that rescale the spatial/temporal dimensions to match the resolutions of a provided target datacube.

While an **OAC1** implementation does not let a user specify a resolution, a client can specify the target number of samples along a dimension, or the scaling factor. While the capabilities of a user-defined process with asynchronous job management go well beyond data, the resampling of a dataset provides a wide range of benefits, the **OAC1** Standard provides a somewhat simpler shortcut for via query parameters of the URL.

The **OAC1** Standard also supports specifying scaling on *any* coverage's dimension, while **openEO** only supports scaling along the spatial plane/dimensions.

3.3.3.12. Tiles

- openEO: -> secondary web services
- OAC1:
 - GET /collections/{collectionId}/coverage/tiles : list the available tilesets
 - GET /collections/{collectionId}/coverage/tiles/{tileMatrixSetId} : tileset description
 - GET /collections/{collectionId}/coverage/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol} : retrieve tile

The **OAC1** Standard, in alignment with the requirements defined in the [OGC API – Tiles Standard](#), supports the retrieval of coverage data in form of tiles. The tiled content is trimmed and resampled to match the boundaries and resolution of a given tile. More than one tileset can be attached to a coverage.

The **openEO** specification does not specify how to provide tiled access directly, but can be easily cascaded by a dedicated secondary tiles service (like [XYZ tiles](#)) thanks to the [/service_types](#) resource.

3.3.3.13. Media encoding

- openEO: [GET /file_formats](#)
- OAC1: [GET /api](#)

Both the **OAC1** and **openEO** specifications provide a list of data formats the server supports, both the data formats that the server can read from (input) and write to (output).

OAC1 embeds the information in the API description available at the [/api](#) endpoint and does not mandate any particular output encoding or format, although CIS JSON is recommended. Binary formats of course are also allowed (CIS RDF, NetCDF, GeoTIFF, PNG, JPEG, JPEG-2000, etc.).

In the **openEO** specification format names and parameters are aligned with the GDAL/OGR formats although custom file formats are also permitted.

3.3.4. References

- **openEO API, v1.2.0** – <https://api.openeo.org/1.2.0/>
- **OGC API – Coverages – Part 1: Core, v0.0.6** – <https://docs.ogc.org/DRAFTS/19-087.html>
- **OGC API – Common** – <https://ogcapi.ogc.org/common/>
- **OGC Abstract Specification Topic 6 – Schema for coverage geometry and functions** – https://portal.ogc.org/files/?artifact_id=19820
- **OGC API – Tiles – Part 1: Core Standard 1.0.0** – <https://docs.ogc.org/is/20-057/20-057.html>
- **OGC Web Coverage Service (WCS) 2.1 Interface Standard** – <http://docs.opengeospatial.org/is/17-089r1/17-089r1.html>

3.4. Crosswalk between the draft GDC standard and the OGC WCPS Standard

The following provides a brief overview of the similarities and differences between the draft GeoDataCube (GDC) API standard and the adopted Web Coverage Processing Service (WCPS) Standard (which also has been adopted by ISO).

References:

- [OGC WCPS standard with WCPS primer](#)
- [ISO IS 19123-3, adopted as update of OGC Abstract Topic 6](#)

3.4.1. WCPS in Brief

The OGC WCPS Standard specifies a GeoDataCube analytics language for server-side evaluation of datacubes modeled as coverages as per the OGC Coverage Implementation Schema (CIS) Standard. WCPS is a [language honed to the datacube structure](#), designed to be compact, as data query languages such as SQL are, while offering specialized datacube functionality. Building a language, rather than an API specification, opened up the flexibility to have queries ranging from simple access and extraction to any complexity of processing. WCPS is high-level and agnostic of any protocols and encodings (such as JSON). WCPS has been shown to work over https GET/KVP, POST, and OAPI.

3.4.2. Comparison of the GDC API and WCPS

Both the [current GDC API draft](#) and WCPS follow the same philosophy of allowing requests ranging from very simple to unlimited complexity made for server-side evaluation. The draft GDC Standard defines a framework which can be utilized as a carrier protocol for manifold extensions, including WCPS, openEO, or general OAPI-Processes. WCPS, on the other hand, is protocol-agnostic and can utilize any carrier, including GET/KVP, XML/POST, SOAP, REST, OAPI-Coverages, OAPI-Processes, and GDC. As such, GDC and WCPS appear complementary.

In terms of functionality, the draft GDC API specification offers access, extraction, scaling, reprojection, and format encoding. For more advanced processing the specification resorts to external mechanisms such as openEO and OAPI – Processes. The rasdaman Testbed-19 GDC implementation maps GDC requests to WCPS queries which then get executed by the rasdaman engine. The same approach is adopted for OAPI-Coverages (and also for WMS and WMTS). Internally, rasdaman translates WCPS into the rasql array query language which, on the side, is [adopted by ISO as part 15 of SQL](#). Bottom line, all OGC coverage API functions can be expressed in WCPS. Additionally, WCPS offers [features going beyond what APIs are considering](#) making it a superset of the functionalities of the various OGC API Standards.

In the diagram below, the draft GDC API and the WCPS Standard are compared. Focus is on “real” datacube operations. OAP I- specific/datacube unspecific concepts (such as collections) have been ignored. The fact that WCPS omits establishing non-datacube concepts allows WCPS to be embedded into any macro model and get coupled with queries in STAC, OAPI – Records, XQuery/XPath, and any other such model.

Further, only the built in mechanisms, not “black boxes” pulled in from other specifications, are considered. The rationale is that the language semantics can greatly help protect from user errors (such as addressing dimensions not existing, taking nulls as real values, etc.) and further allows server-side optimizations.

Table 6

	SPECIFICATION	EXTRACTION	SCALING	REPROJECTION	ENCODING	MANIPULATION	AGGREGATION	FUSION
GDC (openEO)	✓	✓	✓	✓	✓	✓	✓	✓
WCPS	✓	✓	✓	✓	✓	✓	✓	✓

Legend: ✓ = available, ✗ = not available

- through a special micro syntax in the parameters

Aside from functionality, syntax simplicity is an important criterion in practice. Below a relatively simple example is provided: deriving the absolute windspeed from ECMWF ERA5 wind speed u and v components through the common vector formula of $\sqrt{u^2 + v^2}$. In both cases, the color ramp was omitted as it is not of concern for this discussion.

First, the WCPS query:

```
for $c0 in (ERA5)
return
  encode(
    sqrt( pow( $c0.u10, 2.0 ) + pow( $c0.v10, 2.0 ) ) [ time("2000-01-01T01:00:00Z") ],
    "PNG"
  )
```

Figure 1

Next, the openEO code in the Python client describing the same operation:

```
data = connection.load_collection(
  "ERA5",
  spatial_extent = None,
  temporal_extent = ["2000-01-01T01:00:00Z", "2000-01-01T02:00:00Z"],
  bands = ["u10", "v10"]
)
u10 = data.band("u10") ** 2.0
v10 = data.band("v10") ** 2.0
data = u10 * v10
data = data.apply(process = lambda x: sqrt(x))
```

```
result = data.save_result("PNG")
```

Figure 2

In passing, note that WCPS has a formally defined semantics which the current GDC API draft provides through openEO. Such a formalization is useful to ensure that all servers interpret the same query the same way, delivering the same result.

3.4.3. Conclusion

The draft OGC API – GDC Standard follows the openEO language-based approach, and internally chooses a syntax which is easy to parse for machines through process graphs. Client libraries ensure that it is easy to use for humans. That said, both openEO and WCPS fit well as language add-ins to the draft GDC API. WCPS may act as the “glue” to the OAPI – Coverages model.



4

HIGH LEVEL OVERVIEW OF GDC API

HIGH LEVEL OVERVIEW OF GDC API

Based on the pre-requisites provided in the use cases, the crosswalks analysis documented in the previous chapter and the input from the Testbed-19 participants, a first draft of the GeoDataCube API specification was developed. The goal was not to invent a new standard from scratch, but to build on existing specifications and standards that are already widely adopted by the community. To that end, the first GDC API draft was defined based the existing openEO API and STAC API specifications as well as the more mature modules from the OGC API Standards Suite.

Both the openEO and STAC specifications are currently (February 2024) in the process of becoming OGC Community Standards. During the development of the openEO specification and tools, an effort was made to keep aligned as much as possible with the ongoing OGC API Standards work. As a result, there is already considerable alignment in a fashion similar to how STAC work has been following activities related to the OGC API- Features Standard. This means that the proposed draft GDC API specification includes many building blocks from OGC API Standards by design, as well. Specifically, OGC API – Common provides fundamental requirements classes (building blocks). The GDC API description is provided by Common Part 1 as the core requirements class. The data description requirements follows OGC API Common – Part 2: Collections and STAC.

Naturally, a data cube centric model is preferred. Simple and direct data access can be achieved through Coverages and Features endpoints and can be further extended into visualization using openEO secondary Web Services, implementing OGC API Maps, or the traditional W*S suite of standards. Data processing is realized both through openEO as well as OGC API – Processes. Currently the latter are the two most difficult to bring together because each has a completely different design. Ideally, in a final GDC API standard there will be only one processing mode reconciling the two proposed solutions. Specifying both now, however, allows developers to study both solutions in more detail, learn about the strengths and weaknesses of each, and allow usability tests for direct comparison.

Draft Specification - High Level Overview

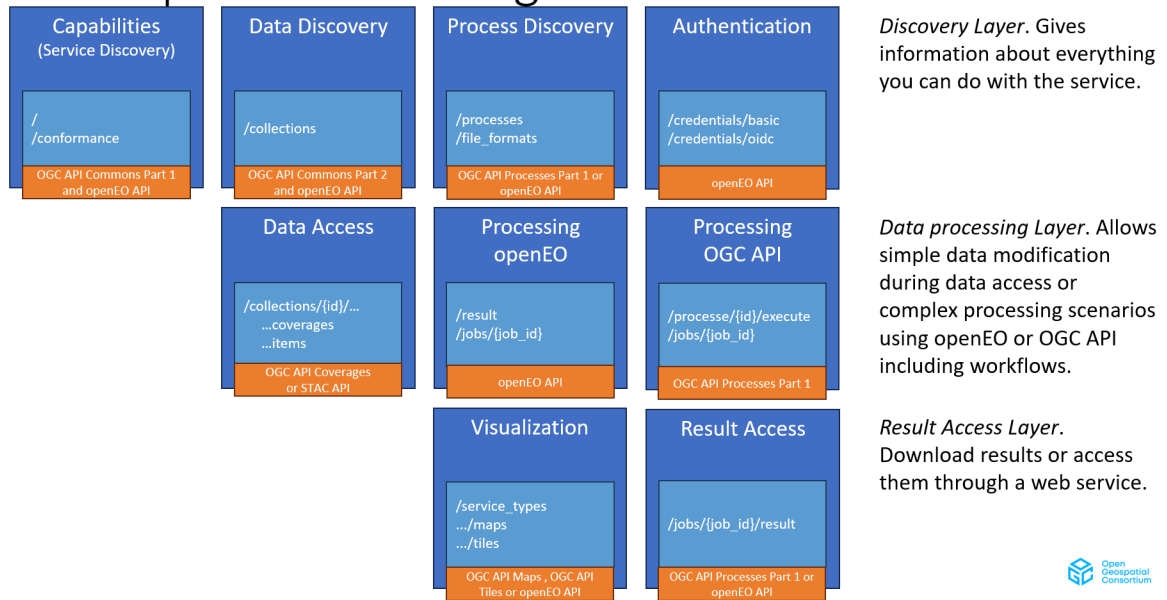


Figure 3 – GDC Draft Specification High Level Overview

4.1. Profiles proposal by Ecere

Ecere’s vision of an OGC GDC API standard is based on profiling existing capabilities defined by other OGC approved and draft standards, such as OGC API – Processes and OGC API – Coverages. This avoids defining yet another way to retrieve data from data cubes, while ensuring that the GDC API definition remains in sync with the still evolving OGC API capabilities.

Different profiles could be defined, regrouping a set of minimal capabilities that must be implemented to conform to each profile.

A “Core” profile could define the minimal capabilities to conform to the GDC API specification, including the ability to retrieve specific fields from a GeoDataCube for a given area, time, and resolution of interest. This corresponds to the OGC API – Coverages – Part 1: Core’s “Core,” “Subsetting,” “Scaling,” and “Field Selection” requirement classes.

A “Processing” profile could define the capability to provide a custom execution request to execute a workflow integrating predefined processes and data available from the API or from any other GDC APIs, while remaining flexible as to the content of that execution request. This would correspond to the “Core” and “OGC Process Description” of OGC API – Processes – Part 1: Core, as well as to the “Collection Input,” “Remote Collection,” and “Collection Output” requirement classes of Processes – Part 3.

Processes – Part 3 “Collection Output” would act as the glue ensuring that any client supporting access to a GeoDataCube using the Core profile, can also access data from a GeoDataCube

generated on-the-fly by a processing workflow using that same Core capability, as defined in *OGC API – Coverages*, with only the additional step of submitting the execution request using an HTTP POST operation, resulting in a virtual GeoDataCube. Accessing such a virtual GeoDataCube from an execution request is already partially supported by the GDAL OGC API driver.

Similarly, the *Processes – Part 3* “Collection Input” and “Remote Collections” would be the glue to allow using data from any OGC GDC API conformant server implementing the Core profile, including those implementing Processing profile, as an input to a GDC processing workflow.

The specific content of the submitted ad-hoc execution requests would be defined by additional profiles, using mechanisms also defined in *OGC API – Processes – Part 3*, including extensions to the *Processes – Part 1* execution requests for “Nested Processes” and “field modifiers” using CQL2 expressions, as well as alternate workflow definition languages such as “Common Workflow Language (CWL)” and “OpenEO process graphs.” Only clients crafting the workflow defined within the request would need to care about the content of the execution requests. Clients accessing and visualizing the resulting GeoDataCube simply submit the request as the request is received.

A “CQL2” Profile could support CQL2 expressions to define filters and derived fields directly as part of data requests, without necessarily implementing the “Processing” profile, but implying support for field modifiers when combined with the “Processing” profile. CQL2 expressions provide a very intuitive and compact way to express simple arithmetic calculations such as vegetation indices, as well as more complex use cases, including several uses cases discussed during the Testbed which were then described using more verbose process graphs.

A separate GDC profile could potentially be defined for the different openEO capabilities based on the proposed openEO OGC Community Standard, which differs in some respects from *OGC API – Processes*.



5

OUTLINE OF ALL CONDUCTED EXPERIMENTS/IMPLEMENTATION

OUTLINE OF ALL CONDUCTED EXPERIMENTS/IMPLEMENTATIONS

5.1. Back-ends

5.1.1. Brockmann Consult – D111/D112 OGC API-GDC instance

5.1.1.1. Introduction

xcube is an open source data cube framework under development by Brockmann Consult GmbH (BC) since 2018. xcube is based on the Python scientific software stack, in particular the xarray package and Zarr storage format. The server component of xcube provides a modular plug-in framework for the implementation of datacube-related web services. Server plug-ins for a number of standard and in-house APIs have already been released, and the Testbed-19 Geodatacube is being implemented as an additional server plug-in.

5.1.1.2. Development and deployment process

Like all xcube code, the GeoDataCube server plug-in is being developed openly at <https://github.com/dcs4cop/xcube> and released under the open-source MIT license. The GeoDataCube features are being added in a series of short-lived branches which are merged with the master branch at regular intervals; releases (usually several per year) are made directly from the master branch as Git tags, GitHub releases, Docker images, and conda-forge packages. The GeoDataCube implementation is thus being automatically integrated into the mainline xcube releases.

5.1.1.3. Public server instance

Our API backend server is at <https://testbed19.api.dev.brockmann-consult.de/api/ogc>. The deployment is updated regularly with the latest codebase from the Testbed-19 development branch, if one currently exists, or the master branch if not.

5.1.1.4. Deploying an xcube server

Since xcube is open source, a server instance can be installed and initialized by any Testbed 19 participant or other interested party. See the xcube documentation at <https://xcube>.

readthedocs.io/ for details on how to install, configure, and run the xcube server. Note that the latest Testbed-19 developments may only be available in a development branch.

The datasets and configuration used in the deployment are also public. To start an xcube server with the same configuration as the BC deployment, run

```
xcube serve --port 8000 --config s3://xcube-viewer-app/testbed19/dev/api/server-config.yaml
```

5.1.1.5. Available datasets

BC has ingested the datasets listed below, as specified in [Use Case B](#) of the OGC Call for Proposals. Ingested data cover at minimum the area of interest defined by the use case.

Table 7

DATASET	COLLECTION LINK	NOTES
Copernicus EU-DEM	EU_DEM_25_E30_N30	Entire E30/N30 tile
Soilgrids250m	soilgrids	
ERA5-Land	era5	2022-2023 only
CMIP6 – SSP2-4.5	cmip	2015-2049 only

Some additional datasets, unrelated to the use case, are also published by the server. The datasets are not listed in the table but can be explored via the Brockmanns GDC instance.

5.1.2. Ecere – GeoDataCube API Instance (D171)

For the server deliverable, Ecere provided a GDC API server instance that supports the OGC API Coverages and Processes capabilities.

End-point: <https://maps.gnosis.earth/ogcapi>

5.1.2.1. Supported Capabilities

The supported OGC API Coverages capabilities include the Core, Subsetting, Scaling, Field Selection, as well as Coverage Tiles. The supported OGC API Processes capabilities include the Core and OGC Process Description. The Ecere implementation also supports additional OGC standards that integrate with the GDC API. They are all listed as follows.

- [OGC API – Coverages](#)

- Subsetting, Resampling, Range subsetting (field selection), filter with CQL2 Expression, derived properties with CQL2 Expressions
- OGC API – Processes
 - Part 1: Core – Synchronous execution
 - Part 3: Workflows & Chaining – Nested/Remote processes; Collection input (local or remote)/output; Field modifiers
- OGC API – Tiles
- OGC API – DGGs (using GNOSISGlobalGrid, ISEA9R and ISEA3H Discrete Global Grid Reference Systems)
- OGC API – Maps
 - Subsetting (Spatial/Temporal/General), Scaling (resampling), CRS

5.1.2.2. Data cubes (collections) of interest

Several data cubes were made available from the GDC API deployment, covering the range of use cases identified for the project:

Climate and weather datasets

- climate:era5:relativeHumidity: Relative humidity (4D with time and pressure, hourly with more than 30 pressure levels, high spatial resolution)
- climate:cmip5:byPressureLevel:temperature: Air temperature (4D with time and pressure level)
- climate:cmip5:singlePressure: Single level climate variables (Precipitation, min/max temperature, surface specific humidity...)
- wildfire:fireDangerIndices: Fire danger indices

Static environmental data

- SRTM_ViewFinderPanorama: Global Elevation (90 meters)
- HRDEM-Ottawa: Ottawa High-resolution DTM (1 meter)
- CHSBathymetryNONNA10: Canadian (East) coastal bathymetry

Categorical coverage (similar use case to land use / land cover maps)

- wildfire:USFuelVegetationTypes: US Fuel Vegetation Types

Remote sensing image time series

- **sentinel2-l2a**: Global sentinel-2 Level-2A imagery (10 meters resolution, B01-B12 bands + B8A + WVP (Water vapor) + SCL (Scene classification map))

For all of the above datasets, `/coverage` will return the raw data values, while `/map` will return an addressable RGB (ARGB) map in some default style (`/styles/{styleId}/map` may return maps in different styles).

Administrative and environmental spatial units (vector maps)

- [Natural Earth \(Physical\)](#)
- [Natural Earth \(Cultural\)](#)
- [OpenStreetMap](#)

These datasets are accessible from the GDC API deployment as *OGC API – Features*, *OGC API – Tiles* (tiled vector feature data and tiled maps), as well as *OGC API – Maps*.

Statistics

- At the time of completing the testbed, the server only supported statistics integrated within spatiotemporal datasets, either as coverage fields or as feature properties.

5.1.2.3. Processes of interest

- **PassThrough**: This process simply cascades an existing collection but supports specifying a `filter` and/or `properties` CQL2 expression to filter, select, or derive fields. See explanations and examples of using *PassThrough* process below to derive new fields from a coverage using various available mechanisms.
- **RFClassify**: This process returns classified fuel vegetation types from sentinel-2 data based on a RandomForest classifier trained on the [USFuelVegetationTypes](#) collection. This process was developed for the Disaster Pilot 2023 Wildland Fire Fuel Indicator Workflow, based on previous efforts from GeoConnections 2020-2021 Modular OGC API Workflow project, [Testbed 17 – GDC API](#), [Testbed 18 – Identifiers for Reproducible Science and Climate Resilience Pilot](#).
- **RenderMap**: This process renders a map of existing collection(s).

NOTE: The output of this process is not a coverage in the true sense in that the *fields* are ARGB values intended to be displayed directly rather than raw values.

5.1.2.4. Addressing Use Case Requirements

Table 8 – Use case A (mapping of capabilities)

REQUIREMENT	CAPABILITY	COMMENTS
ARD spec for EO data product	Schemas (Coverages-1 (Core)) , GeoDataClass	covered in more details by Testbed 19 ARD thread
ARD spec for non-EO data product	Schemas (Coverages-1 (Core)) , GeoDataClass	covered in more details by Testbed 19 ARD thread
Access GDC subset from GDC API implementation	Coverages-1 (Subsetting, Field selection, Scaling)	(supported)
Process GDC content through GDC API for output	Processes-1 , Processes-3 (Collection Input/Output) , Coverages-2 (CQL2 Derived Fields)	(supported)
At least 1 open-source GDC API implementation	N/A	
Some math and band arithmetic operators	Coverages-2 (CQL2 Derived Fields) , Processes-3 (Input/Output Fields Modifiers)	(supported)

Table 9 – Use case B (mapping of capabilities)

REQUIREMENT	CAPABILITY	COMMENTS
Processing of GDC content	Processes-1 , Processes-3 (Collection Input/Output) , Coverages-2 (CQL2 Derived Fields)	(supported)
Interpolation/gap-filling process operators (time +/- space) (output or enrichment...)	Coverages-1 (scaling, slicing sparse dimension permission)	(supported) interpolation for scaling, returning latest available imagery for sentinel-2 data cube when returning 2D format of 2D+time datacube
Inclusion of external processes	Processes-3 (Remote Processes, Remote Collection + Collection Output)	(supported) supported for authorized remote servers
Core attributes / axes x,y, t,v(1...n)	Common-2 (Uniform Additional Dimensions) + common CRS, e.g., EPSG:4326 + Gregorian / UTC time, URIs for additional dimension Units of Measure / semantic	(supported) supporting Lat/Lon for all datasets, time for temporal datasets, pressure for some datasets including multiple pressure levels
AOI Rhine-Meuse Delta	N/A	(supported) several global datasets from demonstration end-point
Data categories EO, stats, land use/land cover, models	Coverages-1 (Core) , Schemas	(supported) EO data available, several other data types
Alternate DGGs support w operators	Processes-3 (Collection Output) , DGGs-1 (Zone Data, Zone Queries) , Coverages-1 (Coverage Tiles)	(supported) DGGs support for GNOSIS Global Grid and ISEA9R – Clarify: what is meant by “alternate DGGs” and “w operators?”

REQUIREMENT	CAPABILITY	COMMENTS
Aggregation operators: generalization, summarization	<u>Coverages-1 (scaling)</u> , <u>Coverages-2 (CQL2 Derived Fields)</u> + Well-known function extensions, e.g., <code>Aggregate((B08-B04)/(B08+B04), 'min', ['time'])</code> (see <i>future work</i> section below)	<i>(in development)</i> Only automatic generalization through Scaling is supported right now. Planning to implement something like <code>Aggregate((B08-B04)/(B08+B04), 'min', ['time'])</code> .
Incorporation of new data sources (client, server, out of band?) transactions/federations/joins?	<u>Processes-3 (Collection Input, Input/Output field modifiers)</u> , <u>Coverages-2 (CQL2 Derived Fields w/ joinCollections)</u>	(supported) Support for remote collections from authorized servers
Outputs: COG, GRIB2, NetCDF, CSV, etc.	<u>Coverages-1 (Core)</u> , <u>Processes-1 (Core)</u> , <u>Processes-3 (Collection Output)</u> and HTTP content-negotiation	(supported) Support for GeoTIFF; next planned format: netCDF

Table 10 – Use case C (mapping of capabilities)

REQUIREMENT	CAPABILITY	COMMENTS
Support many axes	<u>Common-2 (Uniform Additional Dimensions)</u> , <u>Coverages-1 (Core)</u>	(supported) currently providing 4D (lat/lon/time/pressure level) datasets
GDC “subspace” index and query	<u>Coverages-1 (Subsetting)</u> , <u>Coverages-1 (Field selection)</u>	(supported) Subsetting, Field selection supported
Non-rectangular subspaces	<u>Common-2 (Collections)</u> , <u>Coverages-2 (Filtering)</u> + <u>CQL2 Spatial Functions</u> , <u>EDR</u>	(supported) Collections support – Clarify: what is a subspace?
Organization of subspaces (collections, tree)	<u>Common-2 (Hierarchical Collections)?</u>	(supported) hierarchical collections support, planning to soon update to latest proposal – Clarify: what is a subspace?
Axis types: measurable, countable, non-countable	<u>Common-2 (Uniform Additional Dimensions)</u> , <u>Coverages-1 (Core)</u>	(supported)
“Nodata” option for axes (query behavior)	<u>Common-2 (Uniform Additional Dimensions)</u> , <u>slicing sparse dimension permission</u>	(supported) Returning latest available data when returning 2D output format of sentinel-2 Lat/Lon + Time datacube
Multiple time axes (phenomenon, valid?)	<u>Common-2 (Uniform Additional Dimensions)</u> , <u>Coverages-1 (Core)</u>	(supported) Supported in principle, but no example collection
Non-rectangular query (c. f.EDR)	<u>Coverages-2 (Filtering)</u> + <u>CQL2 Spatial Functions</u> , <u>EDR</u>	<i>(in development)</i> Coverages-2 Filtering supported, but Work in progress to implement full CQL2 Spatial operators
Asynchronous execution	<u>Processes-1 (Core: Polling patterns, callbacks)</u> , <u>Processes-3 (Collection Output “on demand” as alternative)</u> <u>Pub/Sub / Async API</u>	Only Processes-3 Collection Output implemented (multiple small synchronous requests as an alternative)

5.1.2.5. Example use of implementation

In this example, use of the Filtering and Derived field extensions, a GeoTIFF coverage is returned from the GNOSIS GDC API implementation for the average Near Surface Air Temperature, converts the temperature from Kelvin to Celsius, and filters out cells where the difference between maximum and minimum is not greater than 10 degrees.

```
https://maps.gnosis.earth/ogcapi/collections/climate:cmip5:singlePressure/coverage?
  filter=(tasmax-tasmin) > 10&
  properties=tas-273.15&
  f=geotiff
```

Figure 4 – OGC API - Coverages request from GNOSIS Map Server using CQL2 expressions to filter cells by values and convert Kelvin to Celsius



Figure 5 – Coverage output of above request, with a color map style applied in QGIS

In this other example, a coverage is returned from the sentinel-2 filtering out cloudy data using the Scene Classification Layer field and computing an Enhanced Vegetation Index from the Near-Infrared, red, and blue bands.

```
https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/coverage?
  properties=2.5 * (B08 / 10000 - B04 / 10000) / (1 + B08 / 10000 + 6 * B04 /
  10000 + -7.5 * B02 / 10000)&
  filter=(SCL >=4 and SCL <= 7) or SCL=11&
  subset=Lat(38.9:39.1),Lon(-4.8:-4.6),time("2017-09-04T11:18:26Z")&
  crs=[EPSG:4326]&
  scale-size=2000,2000&
```

f=geotiff

Figure 6 – OGC API - Coverages request from GNOSIS Map Server using CQL2 expressions to compute an Enhanced Vegetation Index (EVI) and filter out clouds

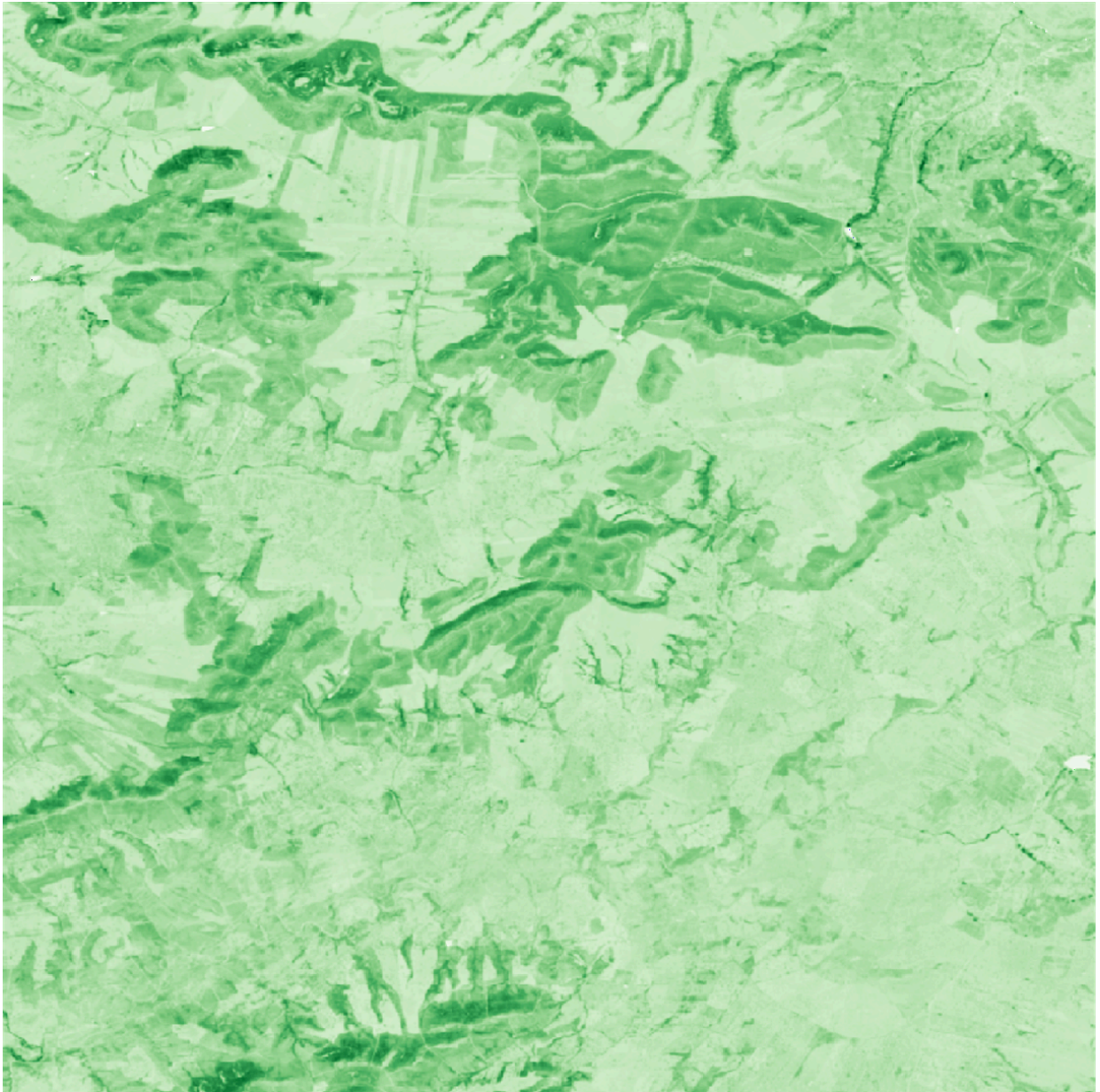


Figure 7 – Coverage output of above request, with a color map styled applied in QGIS

The following is an example using *Processes – Part 3 – “Collection input” / “output”* execution calculating an NDVI from sentinel-2 Level-2A data cube that can be POSTed to

<https://maps.gnosis.earth/ogcapi/processes/PassThrough/execution?response=collection>:

```
{  
  "process" : "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
```



```

    "inputs" : {
      "data" : [
        {
          "collection" : "https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a",
          "properties" : [ "(B08 - B04) / (B08 + B04)" ]
        }
      ]
    }
  }
}

```

Figure 8 – Example PassThrough process execution request (for *Collection Output*)

From the resulting virtual collection one can then request a coverage subset for a particular area, time and resolution of interest:

[https://maps.gnosis.earth/ogcapi/collections/temp-exec-5CB51B82/coverage?subset=Lat\(-16.259765625:-16.2158203125\),Lon\(124.4091796875:124.453125\),time\(\"2022-06-28\"\)](https://maps.gnosis.earth/ogcapi/collections/temp-exec-5CB51B82/coverage?subset=Lat(-16.259765625:-16.2158203125),Lon(124.4091796875:124.453125),time(\)

or a map:

[https://maps.gnosis.earth/ogcapi/collections/temp-exec-5CB51B82/map?subset=Lat\(-16.259765625:-16.2158203125\),Lon\(124.4091796875:124.453125\),time\(\"2022-06-28\"\)](https://maps.gnosis.earth/ogcapi/collections/temp-exec-5CB51B82/map?subset=Lat(-16.259765625:-16.2158203125),Lon(124.4091796875:124.453125),time(\)

(or coverage tiles, or map tiles, or DGGS zone data...) triggering on-the-fly processing.

With authentication, a POST of this execution request to /collections could also automatically create a new collection. The following is an equivalent example using Part 1 – Core synchronous execution (instead of collection input/output) that can be POSTed to

<https://maps.gnosis.earth/ogcapi/processes/PassThrough/execution>.

This example still uses a Part 3 input field modifiers, the "properties" which is at the same level as the "href."

NOTE: The difference with output field modifiers is that the "properties" would be outside and at the same level as "inputs" (and also same level as "process", which is specified in Part 3).

```

{
  "inputs" : {
    "data" : [
      {
        "href" : "https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/coverage?subset=Lat(-16.259765625:-16.2158203125),Lon(124.4091796875:124.453125),time(\"2022-06-28\")&properties=B04,B08&#x26;f=image/tiff;application=geotiff",
        "properties" : [ "(band2 - band1) / (band2 + band1)" ]
      }
    ]
  }
}

```

Figure 9 – Example PassThrough process execution request (for *Synchronous execution*)

Unlike the Collection input/output approach, here the Area/Time/Resolution of interest is implied from those of the “data” input avoiding the need for separate inputs that would specify these, which would be meaningless in Collection Input/Output and would complicate the implementation of processes that support both approaches. Whereas the Collection approach is fully re-usable for any area/time/resolution of interest, this synchronous execution request is only usable once. Note that the names of the bands changed to the default ones that are understood from accessing the GeoTIFF, since GeoTIFF does not currently support encoding band names. With collection input, the fields can be known from the collection schema (to which the Coverages API is transitioning) or currently from the Coverage RangeType. It should also be possible to chain this execution request with the RenderMap process using the Part 3 – Nested process (or Remote Process for the case where the two processes are on different servers) in order to retrieve a rendered map instead of a coverage.

5.1.2.6. Future work: geometry intersections, spatial joins, aggregation and convolution

The planned *OGC API – Coverages – Part – 2: Filtering, deriving fields, aggregation and convolution* extension would define common OGC API building blocks that support filters which values get returned and derived new values using CQL2 expressions.

A Part 2 requirements class could also enable joins with other collections, including feature collections, using a `joinCollections` parameter, facilitating the integration of vector and raster data. The fields from the joined collections would become available for use within the filter and derived field expressions.

An additional example use of the filtering extension would cover different types of queries supported by *OGC API – EDR* use cases, such as trajectories, by using the CQL2 `S_INTERSECTS()` spatial relation function together with a WKT geometry such as a `LineString`. Only the data cube cells matching the function would be returned.

```
filter=S_INTERSECTS(rec.cells, POLYGON((-109 64.39894807, -61.25 64.39894807,
-61.25 76.7652818, -109 76.7652818, -109 64.39894807)))
```

Figure 10 – Filtering using CQL2 polygon geometry

This extension could also define standardized functions allowing the user to perform aggregation along one or more dimensions, using specific aggregating operations such as minimum or average.

```
properties= Aggregate(tasmax, 'max', ['time']),
              Aggregate(tasmin, 'min', ['time']) &
subset=time("2020-01-01":"2020-01-10")
```

Figure 11 – Aggregating on temporal dimension using CQL2 expression

Another standardized function could allow performing convolution, where operations on cell kernels allow the user to easily implement advanced capabilities such as edge detections. This example prototypes defining a Sobel operator used in such use cases:

```
properties=
(
  Convolve(B04, [1,0,-1,2,0,-2,1,0,-1], ['Lat','Lon']) ^ 2 +
  Convolve(B04, [1,2,1,0,0,0,-1,-2,-1], ['Lat','Lon']) ^ 2
) ^ 0.5 &
```

```
subset=Lat(38.9:39.1),Lon(-4.8:-4.6),
time("2017-04-12T11:17:08Z":"2017-09-04T11:18:26Z")
```

Figure 12 – Sobel operator (kernel convolution) implemented as a CQL2 expression

These additional capabilities would likely address a large majority of processing use cases, which can also be achieved with WCPS, openEO, and OGC API – Processes execution requests involving multiple processes, using a more intuitive and concise syntax that can be readily combined with the other building blocks defined in OGC API – Coverages.

5.1.3. Eurac Research – GeoDataCube API Instance

For the server deliverable, Eurac Research provided a GDC API server instance that added OGC API Coverages capabilities to an existing openEO API implementation. The implementation extends the openEO [openeo-spring-driver](#) back-end developed by Eurac Research. The code has been refactored to reduce the dependencies and make it more generalist. It currently supports two data managers: rasdaman and Open Datacube. A new endpoint for OGC API Coverages was added, enabling GET/coverages requests. An added value of this GDC API server instance is the capability to translate OGC Coverages requests to openEO process graphs internally, which can be later re-run again replicating the same request, or to debug using inspecting logs if something went wrong.

End-point: <https://dev.openeo.eurac.edu/>

5.1.3.1. Supported Capabilities

The implemented Coverages API capabilities includes the **Core**, **Subsetting**, and **Field Selection** (from [OGC API – Coverages](#)).

The following request parameters are thus available.

- **bbox**=west,south,east,north for spatial filters
- **datetime**=from[,to] parameter for time trimming (or slicing in case a single timestamp is provided); input timestamp shall be in the format YYYY-HH-MMThh:mm:ssTZ (e.g., 2017-10-31T10:00:00Z); '.' wildcards are supported for open-ended intervals
- **properties**=p1[,p2]* for bands sub-selection, whose names are available in the cube: dimensions/bands section of in the STAC collection document
- **f**=mime for specifying the coverage output format (eg. *application/x-netcdf, image/tiff*)

IMPORTANT

The following exceptions apply to the current implementation.

- The **subset** (and subset-crs) parameter is not accepted, hence subsetting shall be requested through the bbox and datetime parameters.

- **bbox-crs** is not accepted, and lat/lon WGS84 decimal-degrees coordinates are assumed in the bbox parameter.

Further Notes:

- at least a bbox or a datetime filter are required to inhibit download of huge amounts of data (when requested in GeoTiff or NetCDF formats);
- authentication tokens are required in the HTTP request for retrieve a coverage (one can fetch a token by logging in to <https://dev.openeo.eurac.edu/credentials/basic>; send an email to piero.campalani@eurac.edu to get your test credentials); and
- a demo client for creating and submitting process graphs is available at <https://m-mohr.github.io/gdc-web-editor/>.

IMPORTANT

*The **Coordinate Reference System (CRS)** extension has not been implemented, hence no reprojection of output is allowed via **crs** parameter and the coverages are provided in their own native (“storage”) CRS.*

5.1.3.2. Data cubes (collections) of interest

A single data cube was made available from the GDC API deployment. However, the openEO collections available on Eurac Research’s back-end could also easily be enabled to be available via the GDC API implementation.

- **Sentinel-2 L2A**: Only specific tiles covering South Tyrol, Italy. Sentinel-2 Level-2A imagery (10 meters resolution, B01-B12 bands + B8A + WVP (Water vapor) + SCL (Scene classification map))

The /coverage request will return the raw data values.

5.1.3.3. Example use of implementation

A sample GET/coverage request to the Eurac Research server instance is the following:

https://dev.openeo.eurac.edu/collections/s2_l2a/coverage?properties=blue,green,red&datetime=2022-07-01T00:00:00Z/2022-07-05T00:00:00Z&bbox=11.26,46.45,11.40,46.52&f=geotiff

The request specifies an area of interest above the city of Bolzano, where Eurac Research is located. The request selects only the blue, green, and red bands using the properties filtering and stores the result as a GeoTIFF.

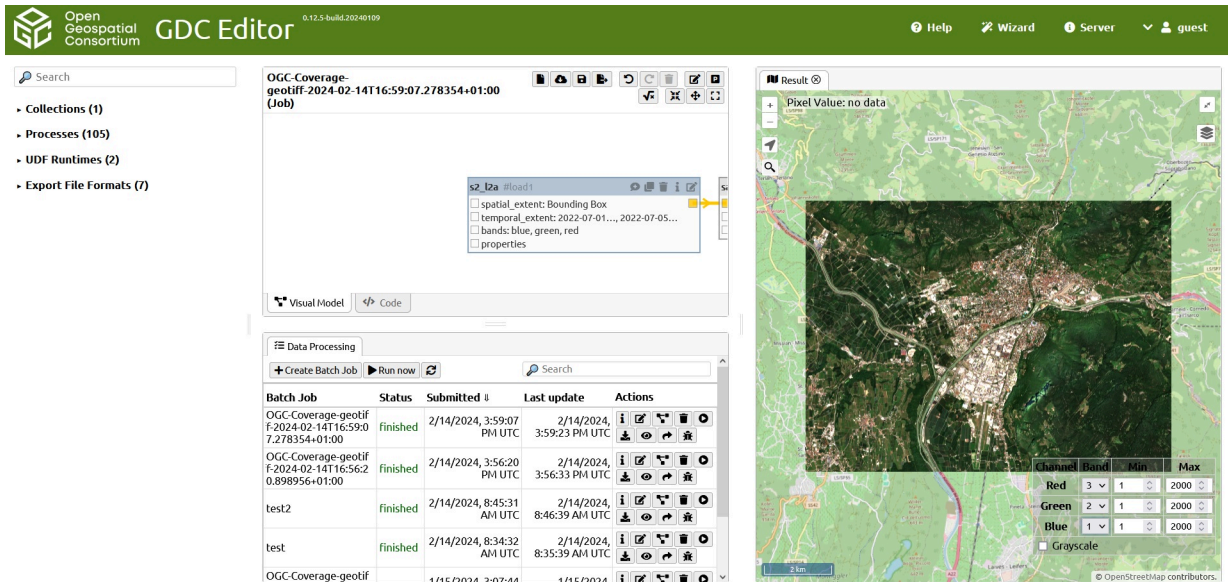


Figure 13 – OGC API - Coverages request from Eurac Research client (GDC Web Editor) to our server, visualized as an RGB composite

5.1.3.4. Future work: GDC API alignment, OGC Processes implementation

Eurac planned to work on the EUMETSAT Use Case, but unfortunately developing the current functionality took longer than expected. Additionally, no implementation for OGC API Processes has been released in Eurac's server yet. Therefore, a possible future work could consist of implementing the OGC API Processes Part:1 standard and trying to harmonize and integrate it with the currently available openEO processes and related workflows encoded in the openEO process graph. This could also be further expanded into a first implementation of OGC API processes part:3.

5.1.4. GeoLabs – Open Source Prototype D111

GeoLabs has implemented two different prototype server instances.

5.1.4.1. ZOO-Project with Deploy, Replace, Undeploy (DRU) support to deploy OpenEO User Defined Processes

- Prototype Server Instance Landing Page: <https://testbed19.geolabs.fr:8720/ogc-api/>
- Service documentation (service-doc relation type): <https://testbed19.geolabs.fr:8720/ogc-api/api.html>

The service documentation contains examples for deploying and executing a simple User Defined Process OpenEO graph.

5.1.4.1.1. Supported Capabilities

- OGC API – Processes
 - [Part 1: Core](#)
 - [Part 2: Deploy, Replace, Undeploy](#)
 - [Part 3: Workflows & Chaining](#) – Nested/Remote processes, deployment of OpenEO graphs using Part 2 draft specification and the openeo conformance class from Part3.

5.1.4.1.2. Description

The first Open Source prototype server implementation provides an environment in which the ZOO-Project, an OGC API – Processes – Part 1 Reference Implementation, is associated with OpenDataCube. The dedicated Docker image is tagged `zooproject/zoo-project:ciab` and is available on DockerHub under the ZOO-Project organization. This tag name was chosen because GeoLabs started with the `cube-in-a-box (ciab)` project. GeoLabs implemented the initial processes for interacting with the OpenDataCube: one (`GDCIndex`) to index new data and another (`GDCList`) to list the available collections.

In addition to Part 1, this version of the ZOO-Project also supports the “OGC API – Processes – Part 2: Deploy, Replace, Undeploy” draft specification. By deploying the ZOO-Project with the DRU support for this study, GeoLabs realized that how implementing the security and relative namespace for searching for processes metadata was irrelevant in that specific case. Consequently, GeoLabs tackled this issue by appending the public namespace (the default location to search for the processes) to the namespace dedicated to user-deployed processes when searching for process metadata. This way, users can access the processes from the public namespace and the process the users have deployed can be stored in the users’ dedicated namespace.

This server instance uses the Part 2 draft specification and the OGC Application package encoding for deploying OpenEO User Defined Processes. The OGC Application Package has a `processDescription` property that contains the metadata information, typically the one retrieved from the `/processes/{processId}` endpoint, when the `executionUnit` is a JSON object with a `format` object with a `mediaType` field having the `application/openEO` value, and as the `value` field the OpenEO UDP graph. Below is a request example for deploying the basic `fahrenheit_to_celsius` OpenEO graph.

```
{
  "processDescription": {
    "id": "fahrenheit_to_celsius",
    "title": "Conversion from fahrenheit to celsius",
    "abstract": "Conversion from fahrenheit to celsius",
    "version": "0.0.1",
    "inputs": {
      "f": {
        "title": "Degrees Fahrenheit",
        "description": "Degrees Fahrenheit",
```


3 nested processes to default the execution mode to asynchronous when invoking a nested process.

5.1.4.2. ZOO-Project with Data Discovery / Access provided by eoAPI and Account Management using Keycloak

- Prototype Server Instance Landing Page: <https://testbed19.geolabs.fr:8717/ogc-api/>
- Service documentation (service-doc relation type): <https://testbed19.geolabs.fr:8717/ogc-api/api.html>

5.1.4.2.1. Supported Capabilities

- OGC API – Processes
 - [Part 1: Core](#)
 - [Part 3: Workflows & Chaining](#) – Nested/Remote processes.
- OGC API – Features
 - [Part 1: Core](#)
 - [Part 3: Filtering and the Common Query Language \(CQL\)](#)
- OGC API – Tiles (Partial support)
 - [Part 1: Core](#)

5.1.4.2.2. Description

The source code for this Prototype Server Implementation is currently available in a ZOO-eoAPI repository under the GeoLabs organization. The source code uses Docker Compose to set up all the required containers and offers a single entry point to multiple (potentially hosted somewhere else) OGC APIs provided by various containers (that may be different hosts or pods depending on the environment which is relied upon). The source code also gives the traditional processing capability offered by the ZOO-Project, with 700+ processes available, while other APIs, such as OGC API – Features and OGC API – Tiles, rely on the different eoAPI platform components.

To provide access to the eoAPI components, GeoLabs added dedicated endpoints to the OpenAPI available per default with the ZOO-Project distributions (Docker images, Docker Compose environments, Helm Charts) by modifying the default `oas.cfg` file, which defines the produced OpenAPI. Then, using the concept of `filter_in` and `filter_out` available in the ZOO-Project, GeoLabs created a process, used as a `filter_in`, dedicated to target requests

coming to a given endpoint and forwarded the request to the defined proxied component to get back the proper response.

This Prototype Server Instance does not require users to authenticate to interact with most endpoints. Nevertheless, GeoLabs partly supports the Account Management and provides access to the `/credentials/oidc` and `/me` endpoints. GeoLabs set up a Keycloak instance on another environment to implement this prototype support. This server offers authentication capability, and the security is very trivial. A user can access every secured endpoint if the user is in the `authorized_users` list or none if not. To get the first endpoint to work correctly, GeoLabs slightly modified the implementation of the ZOO-Project to be able to alter the response that the server will return. Indeed, in this specific case, the response expected by the GDC client applications is partly contained in the content of the `openid-configuration`. As GeoLabs can modify the response content, the `filter_in` process parses the `openid-configuration` file and then creates the desired response with the array containing a single provider, which is the one provided by Keycloak. Only a single provider is supported with the current implementation.

By implementing this basic security support, GeoLabs realized there is no defined way to inform the client application whether or not one endpoint requires authentication. If the OpenAPI can provide such information, GeoLabs could not find any to tell the client that an endpoint does not require authentication. Consequently, GeoLabs faced public and private namespace issues for searching for process metadata described in the previous section.

Also, by deploying this version of the Prototype Server Implementation, GeoLabs realized an issue in the Basic HTML User Interface that comes by default with the ZOO-Project. Indeed, when authentication is on for a given endpoint, as the Basic HTML UI does not provide any way to authenticate, the HTML version cannot load. GeoLabs expects to solve this in the future by integrating JavaScript components within the Basic HTML UI, permitting authentication when required. Nevertheless, as GeoLabs provides access to STAC from the Data Discovery/Access part, the Basic HTML UI is slightly modified to embed the STAC Browser.

Depending on the endpoint used, the media type of the returned content varies which may be text content or binary. The current implementation covers all these cases.

5.1.5. **rasdaman – GDC-API Server Instance**

rasdaman is an Array DBMS with out-of-the-box support for Multi-Dimensional Datacube Management and Analytics. It is available as an open-source community edition as well as an enterprise version suitable for large-scale, federated commercial deployments. Both implement generally the same interfaces, with the query languages based on the ISO SQL/MDA and OGC WCPS standards. Additionally, OGC WMS, WMTS, and WCS are supported. Internally, all requests are mapped to the datacube query language which the server ultimately evaluates. The enterprise edition employs its own query processing engine optimized for scalability and green computing.

5.1.5.1. **GDC-API implementation**

In Testbed-19, the proposed GDC-API were partially implemented in rasdaman community. Specifically:

- Authentication
- Data Discovery / Access
- OGC API – Coverages
- subsetting, resampling, range subsetting
- openEO processes
- predefined and user-defined processes, synchronous process execution

The rasdaman GDC-API implementation is available for access from clients at <https://testbed19.rasdaman.com/rasdaman/gdc>. Implementers of GDC-API clients were provided with access credentials for testing.

5.1.5.2. Data for Use-Case Scenarios

To support the Use-Case scenarios, the Testbed participants imported various data and made the data available through the GDC-API endpoint. The <https://testbed19.rasdaman.com> website provides an overview of the available datacubes.

UC A

- Sentinel-1 GRD, Sentinel-2 L2A, Sentinel-3, Sentinel-5p

UC B

- EU-DEM: A digital surface model (DSM) of EEA39 countries at 25m resolution.
- CLMS 10m: Land use maps at 10m resolution: imperviousness, tree cover, grassland, water, and wetness.
- Soilgrids250m: Global soil property maps at six standard depth intervals with 250m resolution.
- Worldcover: A global land cover product at 10m resolution for 2021 with an accuracy of ~76.7%.
- CMIP6 – SSP2-4.5- EC-Earth3-CC model: Global daily surface-level climate projections covering 2000-2049 with 80km resolution (variables daily maximum near-surface air temperature, daily minimum – near-surface air temperature, near-surface air temperature, near-surface specific humidity, precipitation, and sea level pressure)
- ERA5-Land: Global hourly reanalysis (surface) data for 1982-2013 (variables u10, v10, d2m, t2m, sp, ssrd, tp).

UC C

- ECMWF data regridded to 0.1 grid

5.1.5.3. Results

Mapping the GDC and OAPI-Coverages requests turned out to be straightforward. Only a subset of the query language capabilities was used. WCPS provides a solid basis for the variety of geo datacube APIs that have emerged in the meantime, such as WCS GET, WCS POST/XML, WCS SOAP, OAPI-Coverages, and now additionally the draft GDC API specification. A caveat is that the more recent API specifications convey subtle differences in use and semantics, which required a careful mapping. Here, WCPS can serve as the canonical expression which allows the user to capture and document such differences exactly.

5.1.6. Wuhan University – OGC API-GDC Instance (D111)

Wuhan University successfully implemented an instance of the draft GDC API specification leveraging the GeoCube infrastructure. This infrastructure is deployed across three private cloud servers, utilizing the high-performance computing framework Apache Spark and the distributed storage database HBase. The primary objective was to provide services for the management and analytical processing of heterogeneous data from multiple sources, including raster, vector, and tabular data types.

5.1.6.1. Supported standards or drafts

The following capabilities outlined in the proposed GDC API specification will be implemented:

- OGC API – Common
- STAC
 - CQL2 filtering
- OGC API – Coverages
 - subsetting, scaling, field selection
- OGC API – Processes
 - Part 1: Core – Asynchronous execution mode only
 - Part 3: Workflows and Chaining – Nested local processes and collection output

The endpoint is http://oge.whu.edu.cn/geocube/gdc_api_t19.

5.1.6.2. Available datasets for use-case scenarios

Partial datasets from use case B and use case C were collected. Separate cubes have been established for each data product, incorporating three fundamental dimensions: time, space, and band. Moreover, these cubes can accommodate additional dimensions based on the distinct characteristics of each product.

Use case B

- Copernicus_EU_DEM: The DEM covering the Rhine-Meuse delta region with an original resolution of 25 meters.
- Soilgrids250m: The global digital soil mapping data covering the Rhine-Meuse delta region with an original spatial resolution of 250m. It contains 9 variables including bdod, cec, cfvo, clay, nitrogen, phh2o, sand, silt, and soc. Each variable is available over 6 soil depth intervals: 0-5cm, 5-15cm, 15-30cm, 30-60cm, 60-100cm, and 100-200cm, and only the average value is taken.
- SENTINEL-2 Level-2A MSI: The advanced product derived from the processed multispectral imagery data obtained by the Sentinel-2 satellite with a spatial coverage which includes the Rhine-Meuse delta region, with a spatial resolution of 10 meters and a time range from 2019-01-09 to 2019-04-14.

Use case C

- ECMWF_hsvs&ECMWF_ht3e: The Europe-wide expver hsvs/ht3e data provided by ECMWF has been uniformly sampled to 0.1 degrees x 0.1 degrees. The data contain parameters for Divergence, Geopotential, Relative humidity, Specific humidity, Temperature, U component of wind, V component of wind, Vertical velocity, and Vorticity (relative). The data also include dimensions other than time, space, and band, such as pressure.

5.1.6.3. STAC API implementation

- Basic metadata for all datasets: /collections

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections

The response body contains key attributes as specified by STAC and also includes key attributes required by the OGC API – Common Standard.

- Full metadata for a specific dataset: /collections/{collectionId}

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e

The response body equally contains the key attributes required by both the OGC API – Common Standard and STAC respectively.

- Metadata queryables for a specific dataset: `/collection/{collectionId}/queryables`

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/queryables

All dimensions in the specific cube, apart from the spatial dimension, will be returned as queryable items.

- Fetch items: `/collection/{collectionId}/items`

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/items?filter=pressure=1000

The response body includes metadata information for all scenes in the data cube. This endpoint supports dimension condition filtering via cql2: only cql-text encoding is supported.

- Fetch a specific item: `/collection/{collectionId}/item/{itemId}`

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/items/ht3e_1000_129.128_0

The response body includes metadata information for a single scene and a link for data retrieval.

5.1.6.4. OGC API – Coverages implementation

- Retrieve a coverage: `/collection/{collectionId}/coverage`

Example: [http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/coverage?bbox=0,35,40,70&datetime=2016-10-28T01:01:00Z/2016-10-28T02:01:00Z&properties=Divergence&subset=pressure\(1000\)&f=tif](http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/coverage?bbox=0,35,40,70&datetime=2016-10-28T01:01:00Z/2016-10-28T02:01:00Z&properties=Divergence&subset=pressure(1000)&f=tif)

The endpoint supports retrieving data from the data cube and returning the data in either GeoTIFF or netCDF format. The endpoint also supports query parameters such as `bbox`, `datetime`, `properties`, `subset`, `scale-size`, `scale-axes`, `scale-factor`, and `format` (`tif`, `netCDF`) to return the data relevant to the area of interest.

- Retrieve the domainset of a coverage: `/collection/{collectionId}/coverage/domainset`

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/coverage/domainset

The endpoint returns the domainset of the coverage, which includes all dimensions of the cube except for the band dimension.

- Retrieve the rangetype of a coverage: `/collection/{collectionId}/coverage/rangetype`

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/collections/ECMWF_ht3e/coverage/rangetype

The endpoint returns the rangetype of the coverage, which includes information about the band (or variable) dimension.

5.1.6.5. OGC API – Processes implementation

- Retrieve the list of all predefined processes: /processes

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/processes

This implementation offers a series of predefined processes, including data loading (loadCube), mathematical operations (add, subtract, divide, normalize), and aggregation operations (aggregate), among others.

- Retrieve the description of a specific process: /processes/{processId}

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate

- Execute a process: /processes/{processId}/execution

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/execution

Request body example:

```
{
  "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate",
  "inputs": {
    "data": {
      "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/normalize",
      "inputs": {
        "data": {
          "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/loadCube",
          "inputs": {
            "cubeName": "SENTINEL-2 Level-2A MSI",
            "extent": "4.7,51.7,4.85,51.8",
            "startTime": "2019-01-17 00:00:00",
            "endTime": "2019-01-20 00:00:00"
          }
        },
        "dimensionName": "bands",
        "dimensionMembers": ["B8", "B3"]
      }
    },
    "dimensionName": "time",
    "method": "mean"
  }
}
```

Figure 15

The endpoint supports local processes nesting and exclusively operates in asynchronous execution mode, creating a job upon execution.

- Retrieve the list of all jobs: /jobs

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/jobs

The endpoint returns all existing jobs.

- Retrieve the specific job: /jobs/{jobId}

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/jobs/34b59622-0faf-4fb2-8029-1fe824d7f51b

The endpoint returns the information about a specific job, including the execution status.

- Retrieve the job results: /jobs/{jobId}/results

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/jobs/34b59622-0faf-4fb2-8029-1fe824d7f51b/results

The endpoint returns the execution result of the process.

- Execute a process with the collection output: /processes/{processId}/execution?response=collection

Example: http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/execution?response=collection

Request body example:

```
{
  "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate",
  "inputs": {
    "data": {
      "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/normalize",
      "inputs": {
        "data": {
          "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/loadCube",
          "inputs": {
            "cubeName": "SENTINEL-2 Level-2A MSI"
          }
        },
        "dimensionName": "bands",
        "dimensionMembers": ["B8", "B3"]
      }
    },
    "dimensionName": "time",
    "method": "mean"
  }
}
```

```
}
```

Figure 16

The response will redirect to a description document of a collection. This service supports triggering computations through the retrieval mechanism of OGC API – Coverages and provides the computation results.

Example:

5.2. Clients

5.2.1. Ecere – Visualization Client (D173)

As a primary deliverable, Ecere developed a visualization client based on Ecere’s GNOSIS Cartographer 3D geospatial visualization tool.

Ecere improved the GNOSIS Cartographer software, a desktop 3D visualization client, as well as the underlying GNOSIS library, to better interoperate with the different participants API endpoints, to retrieve and process GDC data.

For example, with multi-dimensional stores such as CMIP5 data for wind velocity, which includes an atmospheric pressure dimension, the client supports changing the pressure level for selecting the slice of the atmosphere to visualize.

GNOSIS Cartographer includes a workflow editor which can build a workflow by discovering processes and collections available from a GDC API, assemble the workflow into an execution request, and visualize the results.

5.2.1.1. Supported capabilities

The following are the supported OGC API capabilities:

- [OGC API – Coverages](#) (including support for “Coverage Tiles”)
- [OGC API – Processes](#) (including support for “Collection Output”)
- [OGC API – Tiles](#)
- [OGC API – Maps](#)

The capabilities based on openEO were not implemented for this client.

5.2.1.2. Example use of implementation

The following image demonstrates visualizing CMIP5 data for wind velocity retrieved from the Copernicus Data Store. In addition to the spatial and temporal dimension, this datacube includes an atmospheric pressure dimension. The client supports changing the pressure level for selecting the slice of the atmosphere to visualize.

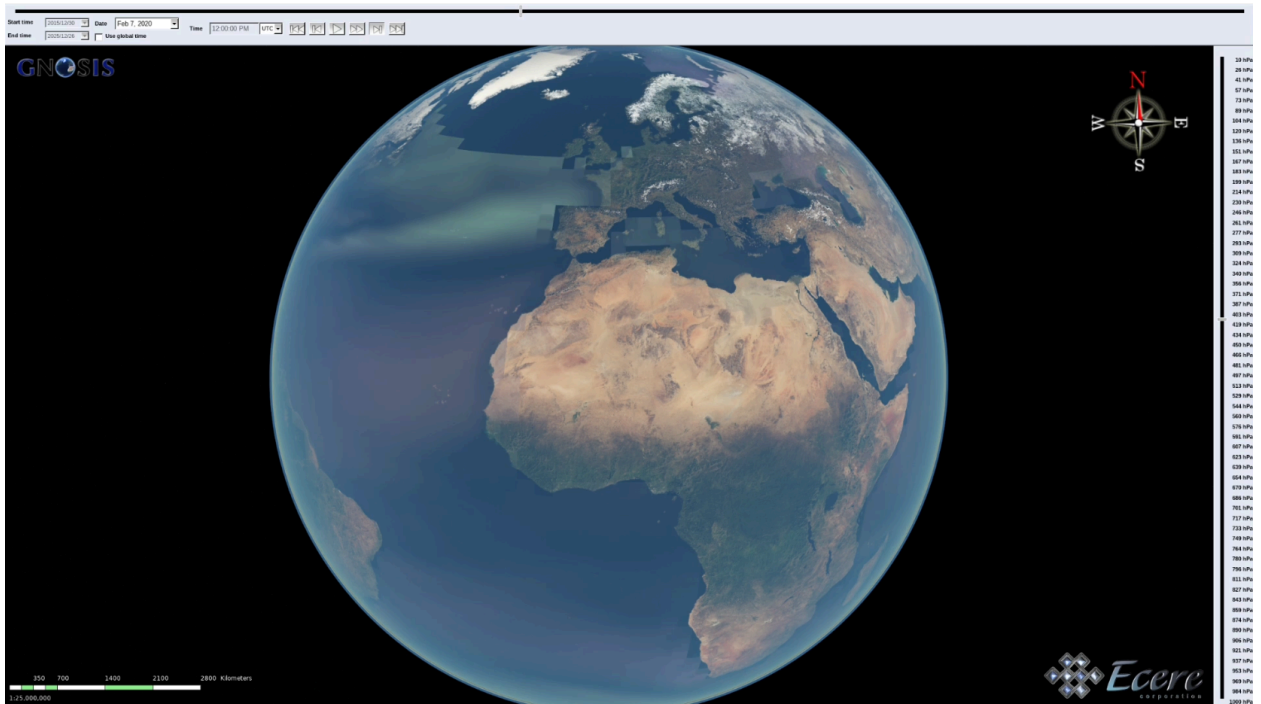


Figure 17 – Cartographer visualizing CMIP5 pressure and wind velocity

The following image shows a visualization of ERA5 relative humidity data which was also retrieved from the Copernicus Data Store. This dataset also includes an atmospheric pressure dimension, with a finer granularity of pressure levels. The spatial and temporal resolution is also higher, whereas only a small subset for six days was loaded onto the server.

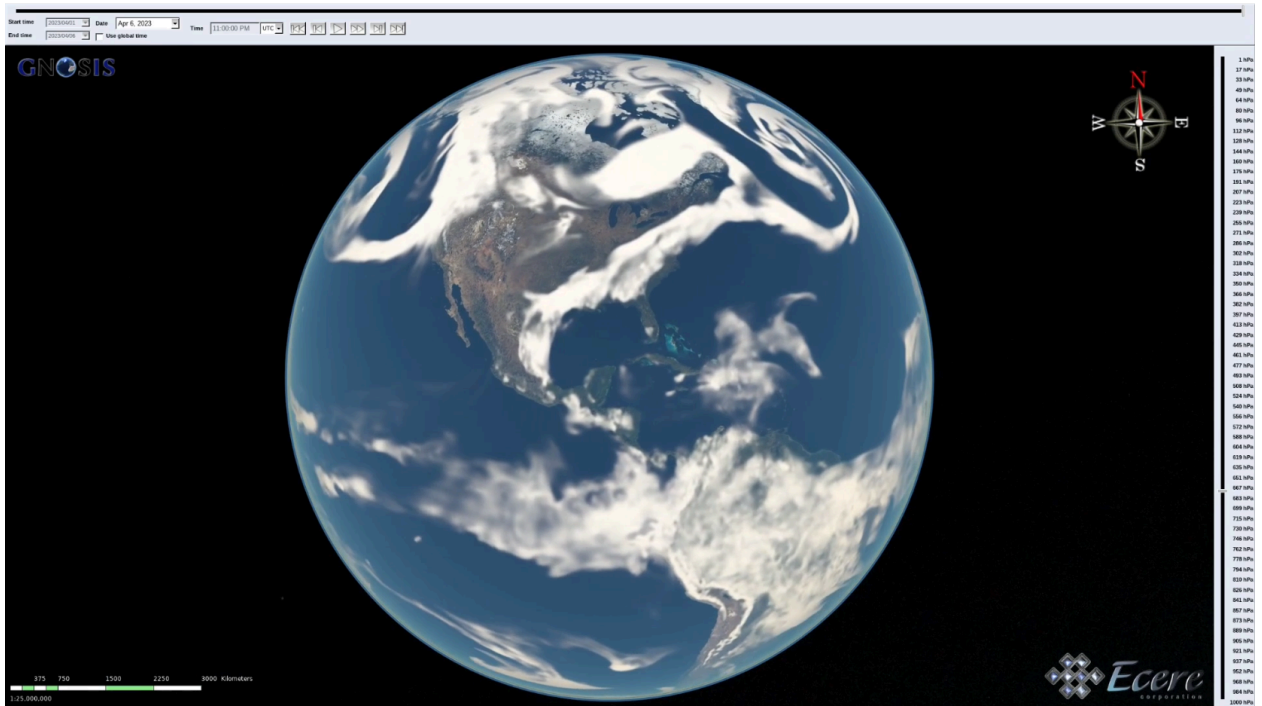


Figure 18 – Cartographer visualizing relative humidity

5.2.2. Eurac Research – GDC Web Editor

The GDC Web Editor is based on the openEO Web Editor, which is the main graphical user interface to interact with openEO. The GDC Web Editor is a web application that allows users to interactively explore a GDC API implementation, create and execute processes, and manage jobs. Additional functionality is available for servers that support the openEO API specification.

For the GDC Web Editor two components of the openEO JavaScript ecosystem had to be adopted:

1. [openEO JavaScript Client](#); and
2. [openEO Web Editor](#).

The openEO JavaScript client is used to communicate with GDC API endpoints. The GDC Web Editor is still primarily an openEO client and as such works internally based on the openEO architecture and concepts. This means internally all requests are translated into GDC API requests and the responses are translated back into openEO responses. This translation work is done by the openEO JavaScript client, which has a new Migration class for GDC API responses and requests. As the draft GDC API standard is mostly a combination of OGC API – Coverages, OGC API – Processes, and openEO, this migration is basically a generic converter between the OGC APIs and openEO.

The added functionality in the GDC Web Editor includes:

- exposing which collections are Coverages and which are openEO data cubes;
- visual discovery of OGC API – Coverages endpoints;
- visual discovery of OGC API – Processes endpoints;
- synchronous and asynchronous execution of workflows for OGC API – Processes (Part 1 and a small subset of Part 3);
- visual process chaining in the model builder for OGC API – Processes, including ability to set parameters through an easy form;
- (Batch) Job management for OGC API – Processes; and
- coverage Download via the Wizard functionality.

The conversion between openEO and OGC API implementations was relatively painless. During development the following difficulties were encountered.

- The OGC API Processes Standard specifies using return values to describe a choice between multiple output file formats. openEO follows a different architecture where the file format is a parameter of the `save_result` process. This means every OGC API Process receives a new parameter format which users can fill. Unfortunately, some implementations require that for every process in a chain, which is not very user-friendly and not very intuitive.
- openEO uses a more fine-grained schema for process parameters and return values, which offers a much better user experience in a UI to fill parameters. This means that in an openEO implementation the client can offer, for example, a bounding box selector. However, in an OGC API – Processes implementation the parameters must be provided as defined in the process schema (e.g., array of numbers or a string of comma-separated values).
- The data cube concepts in the OGC API – Coverages Standard and openEO specification differ slightly and use different terminology, which cannot be translated 1 to 1. Also, the terminology differences are exposed to the user, which is confusing.
- Some functionality that exists in an openEO implementation is not available in OGC API – Processes implementation. As such, the user experience is less good for OGC API – Processes server implementations.
- The weakest point of a client OGC API – Processes implementation is constructing workflows that consist of multiple processes. While there is basic support for it, the approach is not very user-friendly and various parts of OGC API – Processes – Part 3 cannot be encoded.

The source code of the GDC Web Editor and the adapted openEO JS client are available as open-source software, released under the Apache 2.0 license. The source code and a demo version can be found here:

- The source code of the GDC Web Editor is available at <https://github.com/m-mohr/gdc-web-editor>
- The source code of the adapted openEO JavaScript client is available at <https://github.com/Open-EO/openeo-js-client/tree/gdc-api>
- The hosted demo version of the GDC Web Editor is available at <https://m-mohr.github.io/gdc-web-editor/>

Generally, a client that can interact with the GDC API endpoints can be implemented without major effort, including support for OGC API – Processes and OGC API- Coverages.

An exemplary approach at converting between the different APIs and entities is shown in the openEO JavaScript client. The interoperability experiments show good compatibility across various servers. All use cases in Testbed 19 can be executed with the client assuming that the server supports the required functionality.

5.2.3. Fengchia University(GIS.FCU) – Data Client (D113)

- Dev. environment
 - demo site
 - link: <https://tm.gis.tw/testbed19client/>
- Testing example
 - ecere
 - test layer : SRTM_ViewFinderPanorama

```
{
  "process": "https://maps.gnosis.earth/ogcapi/processes/RenderMap",
  "inputs": {
    "background": "navy",
    "transparent": false,
    "layers": [
      {
        "href": "https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage?subset=Lat(21.8:25.4),Lon(120:121.9)%26f=image/tiff"
      }
    ]
  }
}
```

Figure 19

- rasdaman(openEO)

- test layer : ERAS

```

{
  "process_graph": {
    "load1": {
      "process_id": "load_collection",
      "arguments": {
        "id": "ERA5",
        "spatial_extent": null
      }
    },
    "subset1": {
      "process_id": "subset",
      "arguments": {
        "data": { "from_node": "load1" },
        "subset": [ { "dimension": "time", "lower": "\"2000-01-01T01:00:00Z\""
} ]
      }
    },
    "subset4": {
      "process_id": "subset_band",
      "arguments": {
        "x": { "from_node": "subset1" },
        "y": "u10"
      }
    },
    "subset5": {
      "process_id": "subset_band",
      "arguments": {
        "x": { "from_node": "subset1" },
        "y": "v10"
      }
    },
    "power6": {
      "process_id": "power",
      "arguments": {
        "x": { "from_node": "subset4" },
        "y": 2.01
      }
    },
    "power7": {
      "process_id": "power",
      "arguments": {
        "x": { "from_node": "subset5" },
        "y": 2.01
      }
    },
    "add8": {
      "process_id": "add",
      "arguments": {
        "x": { "from_node": "power6" },
        "y": { "from_node": "power7" }
      }
    },
    "sqrt9": {
      "process_id": "sqrt",
      "arguments": {
        "x": { "from_node": "add8" }
      }
    },
    "save10": {

```

```

    "process_id": "save_result",
    "arguments": {
      "x": { "from_node": "sqrt9" },
      "format": "PNG",
      "options": "{\\\\"colorMap\\\\"": { \\\\"type\\\\"": \\\\"intervals\\\\"", \\\\"
\\colorTable\\\\"": { \\\\"1500\\\\"": [121, 145, 171, 255], \\\\"2500\\\\"": [125,
255, 209, 255], \\\\"3500\\\\"": [171, 254, 255, 255], \\\\"5000\\\\"": [114, 224,
176, 255], \\\\"8000\\\\"": [116, 224, 87, 255], \\\\"9000\\\\"": [192, 255, 48,
255], \\\\"10000\\\\"": [254, 255, 0, 255], \\\\"11000\\\\"": [255, 255, 135, 255], \\\\"
\\\\"12000\\\\"": [245, 218, 98, 255], \\\\"13000\\\\"": [255, 150, 51, 255], \\\\"
\\\\"14000\\\\"": [255, 102, 102, 255], \\\\"15000\\\\"": [254, 15, 255, 255], \\\\"
\\\\"16000\\\\"": [203, 0, 204, 255] } } }"
    },
    "result": true
  }
},
"parameters": []
}

```

Figure 20

- whu

```

{
  "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate",
  "inputs": {
    "data": {
      "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/
normalize",
      "inputs": {
        "data": {
          "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/
processes/loadCube",
          "inputs": {
            "cubeName": "SENTINEL-2 Level-2A MSI",
            "extent": "4.7,51.7,4.72,51.72",
            "startTime": "2019-01-17 00:00:00",
            "endTime": "2019-01-20 00:00:00"
          }
        },
        "dimensionName": "bands",
        "dimensionMembers": ["B8", "B3"]
      }
    },
    "dimensionName": "time",
    "method": "mean"
  }
}

```

Figure 21

- Geolabs

```

{
  "inputs": {
    "il": [
      {
        "href": "http://geolabs.fr/dl/Landsat8Extract1.tif"
      }
    ],
  }
}

```

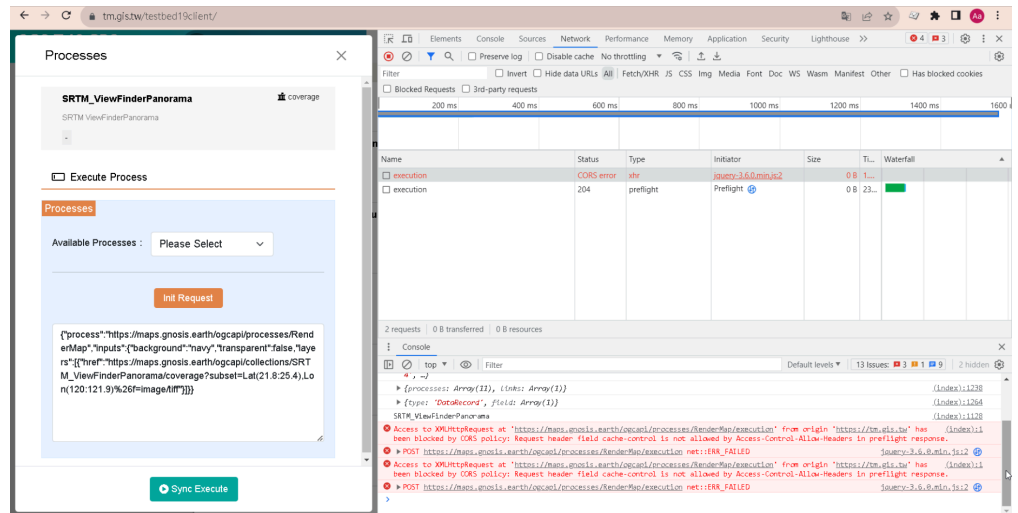
```

    "out": "float",
    "exp": "im1b3,im1b2,im1b1",
    "ram": 256,
    "layers": [
      {
        "href": "Lat(3.4720895754867920:3.8112262152018426),Lon(43.276499596965
1096:43.5081449510672016)"
      }
    ]
  },
  "outputs": {
    "out": {
      "format": {
        "mediaType": "image/jpeg"
      },
      "transmissionMode": "reference"
    }
  },
  "response": "raw"
}

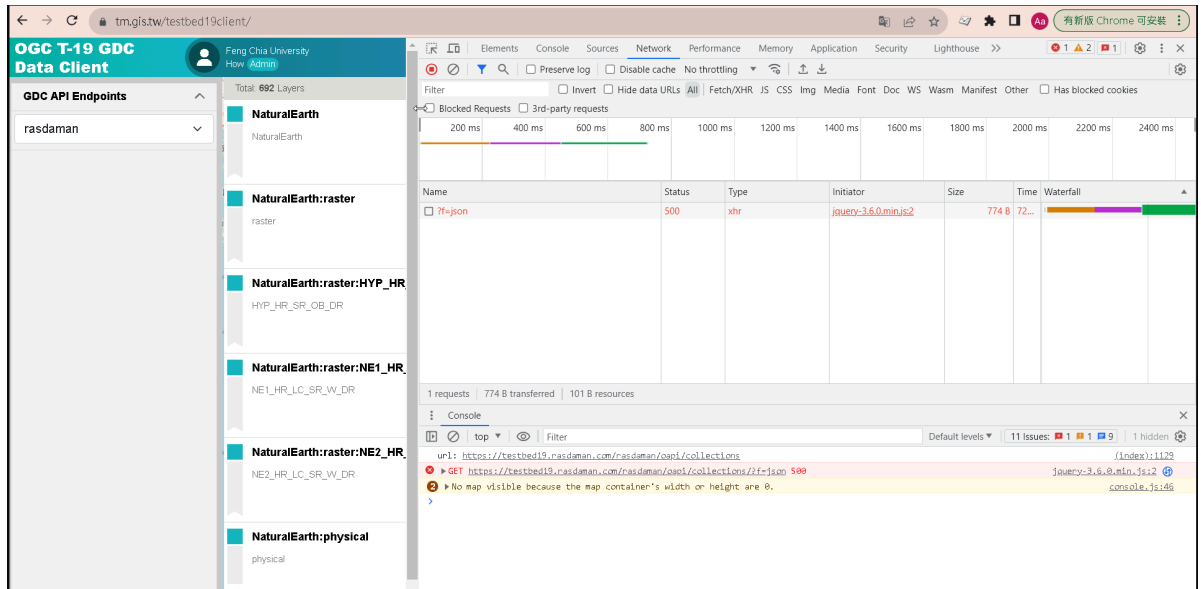
```

Figure 22

- TIE issues
- CORS
 - This is a common issue when integrating with third-party endpoints on the frontend because security typically does not allow direct integration like this. The standard practice is to use a backend service to replace the frontend in accessing the service.
 - Ecere GeoDataCube API implementation.
 - post

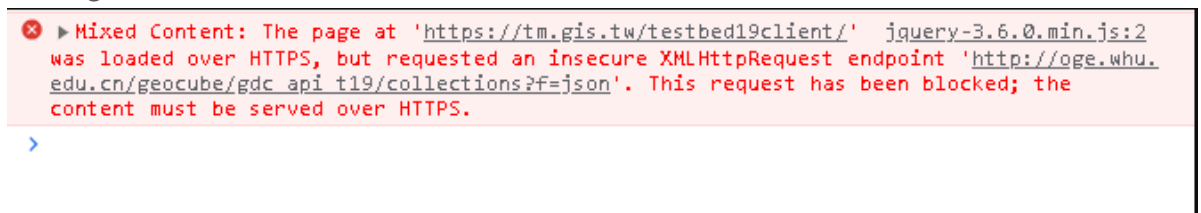


- get request with basic authentication:



- Chrome no longer recommends using HTTP.
 - However, there are still some endpoints found to be providing data through HTTP. If the data is generally public, it might not be a concern. However, when transmitting sensitive or high-security imagery in this manner, there is a risk of the transmission being intercepted. Additionally, most modern browsers now default to disallowing HTTP. Even if an HTTP address is entered, the address will be forcibly redirected to HTTPS for the request. Of course, developers can still interact with HTTP through backend services. It is recommended, whenever possible, to provide services using HTTPS for better security.

- message from browser:



- Async Processes/Execution
 - During implementation of the client, it was observed that computations can be performed in both synchronous and asynchronous modes. For client implementers, consideration must be given to handling and presenting results for both types of operations. For example, in synchronous operations, results can be directly displayed on the map or converted into downloadable files. In asynchronous operations, a jobID is obtained first and the frontend needs to employ a polling mechanism to check the execution progress of the job. When the computation is complete and results are available, the results can then be presented on the map or made available for file download. Alternatively, other notification methods such

as push notifications, LINE, Telegram, Email, etc., can be integrated to inform users when the job is finished.

- Ecere
 - <https://maps.gnosis.earth/ogcapi/processes/actionName/execution>
- Rasdaman
 - <https://testbed19.rasdaman.com/rasdaman/openeo/result>
- whu
 - The approach supporting only asynchronous JobID may pose challenges in presenting results promptly on the map. Would it be acceptable to provide a download option instead?
 - The processes were divided into the following three steps.
 - http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/jobs/3257bfe9-1c3b-4331-90c7-686c0018f0d3
 - http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/jobs/3257bfe9-1c3b-4331-90c7-686c0018f0d3/results
 - http://oge.whu.edu.cn/api/oge-data/data/gdc_api/3257bfe9-1c3b-4331-90c7-686c0018f0d3/Coverage_2023_09_05_13_58_21.tif

- Async processes message from the browser:

http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/execution

POST http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/execution

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

7     "data": {
8         "process": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/loadCube",
9         "inputs": {
10            "cubeName": "SENTINEL-2 Level-2A MSI",
11            "extent": "4.6819,51.6876,4.8784,51.8346",
12            "startTime": "2019-01-17 08:59:59",
13            "endTime": "2019-01-19 21:00:01"
14        }
15    },
16    "dimensionName": "bands",
17    "dimensionMembers": ["B8", "B3"]
18  },
19  },
20  "dimensionName": "time",
21  "method": "mean"
22  }
23  }
24  }

```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

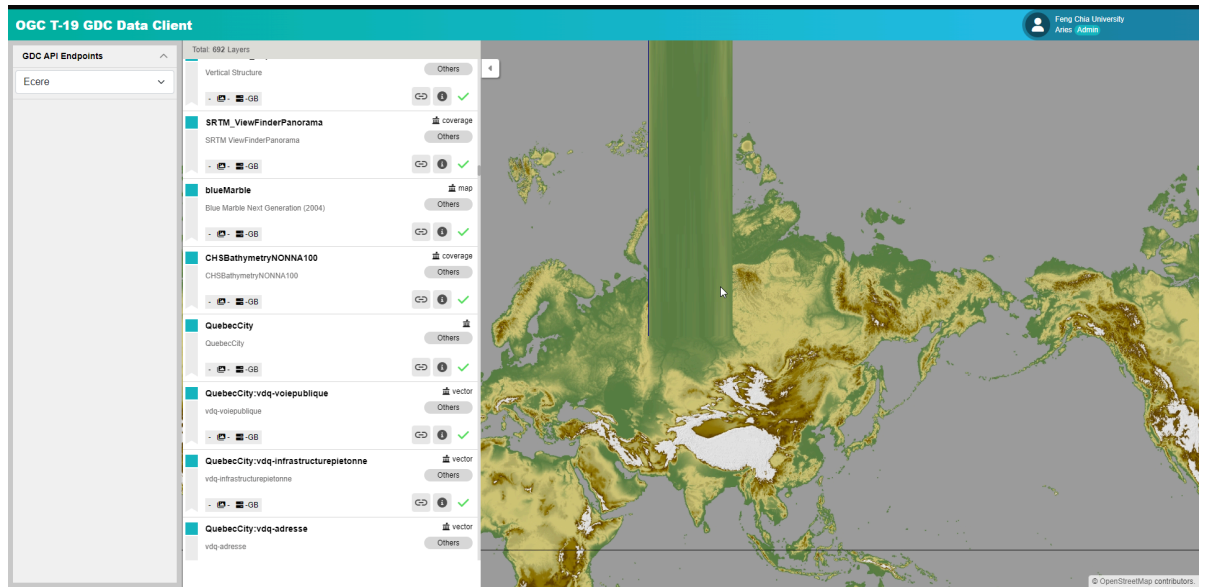
1  {
2    "jobID": "3257bfe9-1c3b-4331-90c7-686c0018f0d3",
3    "status": "accepted",
4    "message": "Process started",
5    "progress": 0,
6    "created": "2023-09-05T13:58:15.235",
7    "links": [
8      {
9        "title": "aggregate",
10       "rel": "self",
11       "type": "application/json",
12       "href": "http://oge.whu.edu.cn/geocube/gdc_api_t19/processes/aggregate/jobs/3257bfe9-1c3b-4331-90c7-686c0018f0d3"
13     }
14   ]
15 }

```

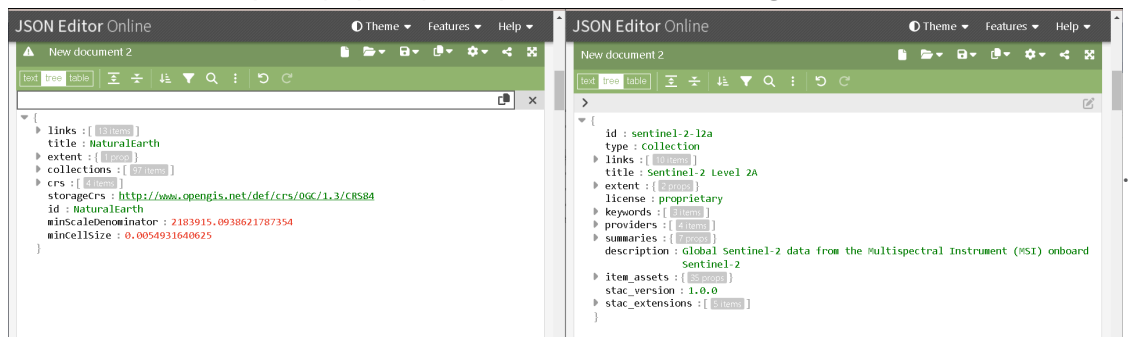
- Front-end library issue

- This is an issue related to coordinate transformation in OpenLayers. OpenLayers typically uses map projection coordinate systems (CRS) such as EPSG:3857 (Web Mercator) or other equidistant projection coordinate systems. The ranges of these coordinate systems are limited to a finite extent, with latitudes approximately between -85.0511 and 85.0511. When attempting to display latitudes beyond this range on the map, it may result in abnormal image rendering, as these projection coordinate systems do not support polar regions or areas too far from the poles. If there is need to display polar regions or geographical data beyond these ranges in OpenLayers, a different projection and coordinate system may need to be used, such as EPSG:4326 (WGS 84 latitude-longitude coordinate system) or another coordinate system that suits the requirements. In these coordinate systems, latitudes can exceed 90 degrees (North Pole) or go below -90 degrees (South Pole), and the map can accurately display these regions.

- Part of the coverage exposed an inaccurate shape:



- Processes format inconsistency
 - In other Endpoints, the 'process' is included in the format for the endpoints developed, utilizing a unified endpoint to perform computations based on the specified 'process' within the format. However, GeoLabs operates differently by having distinct endpoints based on the 'process,' and the format does not explicitly specify the 'process'. left:ecere; right:Geolabs:



5.2.4. Geomatys – Data Client (D173)

The Geomatys client uses the Unreal Engine 3D game engine. For all geospatial aspects, 3D terrain, and tile management, the Cesium Unreal plugin developed by Cesium is relied on. Changes were made to the Cesium plugin to enable time management (e.g., temporal WMS), Map Tiles querying, and other useful improvements for Testbed-19.

Using Unreal's blueprints and C++ support, Geomatys developed the various classes and tools needed to implement the Testbed-19 OGC GDC APIs (see below for capabilities supported by the client). Geomatys was also able to develop several ways of displaying data. On the one hand, raster data can be requested via the Maps API (only maps tiles), which can then be displayed on the Cesium-generated terrain. On the other hand, data from the Coverages API can be displayed

in different ways. (Geomatys products do not support coverages tiles.) Thanks to a “PostProcess” system, the user can choose how to display data from Coverages.

For example, a user can request ECMWF data from a server that provides such data. The user selects the region (lat/lon) of interest, the desired altitudes (pressures), and the moment in time (2023-06-11T00:00:00.000Z for example). The user can also choose to request only the U and V bands for wind. A display service is then created (PostProcess) that manages the wind display. The bands to be queried are assigned to the service, then once the query is done, the wind data is displayed in the client.

For this post processing system, wind, humidity, and temperature data are currently supported.

In terms of technical functionalities, OGC API building blocks were integrated as described below: coverage subsetting (spatial and temporal), coverage field selection (select specific bands in a coverage), and coverage scaling.

For the moment, the Geomatys system only handles coverages in GeoTIFF format. However, using the gdal library to decode files, other formats will be added in the future.

In the rest of this section, the OGC API Standards (building blocks) supported by the client are now described.

Supported capabilities :

Geomatys supports OGC API endpoints (not OpenEO or STAC API)._

- **OGC API – Collections** ([Common-1 \(Core,Landing Page\)](#), [Common-2 \(Collections\)](#))
- **OGC API – Coverages** (subsetting, scaling, field selection) ([Coverages-1 \(Core\)](#), [Coverages-1 \(Subsetting\)](#), [Coverages-1 \(Scaling\)](#), [Coverages-1 \(Field Selection\)](#)) ✓ (only with png/jpg/geotif and not tiled)
- **OGC API – Maps** ([Maps-1 \(Core\)](#)) ✓ (only tiled rasters)

5.2.5. Wuhan University – Data Client (D113)

Leveraging the online spatiotemporal computing cloud platform, [Open Geospatial Engine \(OGE\)](#), developed by Wuhan University, the Wuhan Testbed 19 participants designed and developed an OGE-DataClient that allows users to explore GDC through a visual interface online. OGE-DataClient adheres to the draft GDC API specification draft proposed in this Testbed. The front-end is developed in TypeScript, and automatically assembles requests and parses results based on the agreed-upon OpenAPI specification. For data retrieval, including retrieving Coverage and executing Process results, requests are assembled in the backend service. Then the resulting data is appropriately formatted for seamless display in the Cesium 3D map component. Ideally, the client can connect to any endpoint that adheres to the agreed-upon GDC API specifications, providing capabilities for cube data retrieval and processing. However, it is important to note that, as mentioned in the following sections, this client does not support all the capabilities specified in the draft GDC API specification draft. Further improvements and additions to functionalities will be made in the future.

The demonstration link is: <http://oge.whu.edu.cn:8023/ogc-data-client>

The client focuses on the following features of interest.

- Retrieving all available cube data and processes under a specified GDC API endpoint by entering the login page link.
- Viewing all dimension information for each cube.
- Supporting data retrieval with field selection and performing subset operations along any dimension.
- Supporting map visualization of data in GeoTiff and Png formats.
- Supporting synchronous or asynchronous execution of processes compliant with the OAP specification and synchronous execution of OpenEO processes.
- Visualizing processing results on a map.
- Retrieving and visualizing results from a virtual collection compliant with the OAP-3 specification, executing workflow on demand.

5.2.5.1. Supported approved standards and drafts standards

- OGC API – Common
- OGC API – Coverages
 - subsetting, field selection
- OGC API – Processes
 - Part 1: Core
 - Part 3: Workflows and Chaining – Nested processes and collection output
- OpenEO API
 - Synchronization only

5.2.5.2. Implementation

OGC API – Coverages

The OGE-DataClient automatically generates corresponding filtering conditions by parsing the metadata information of the data cube, including rangetype and domainset. Users can retrieve subsets of data as needed.

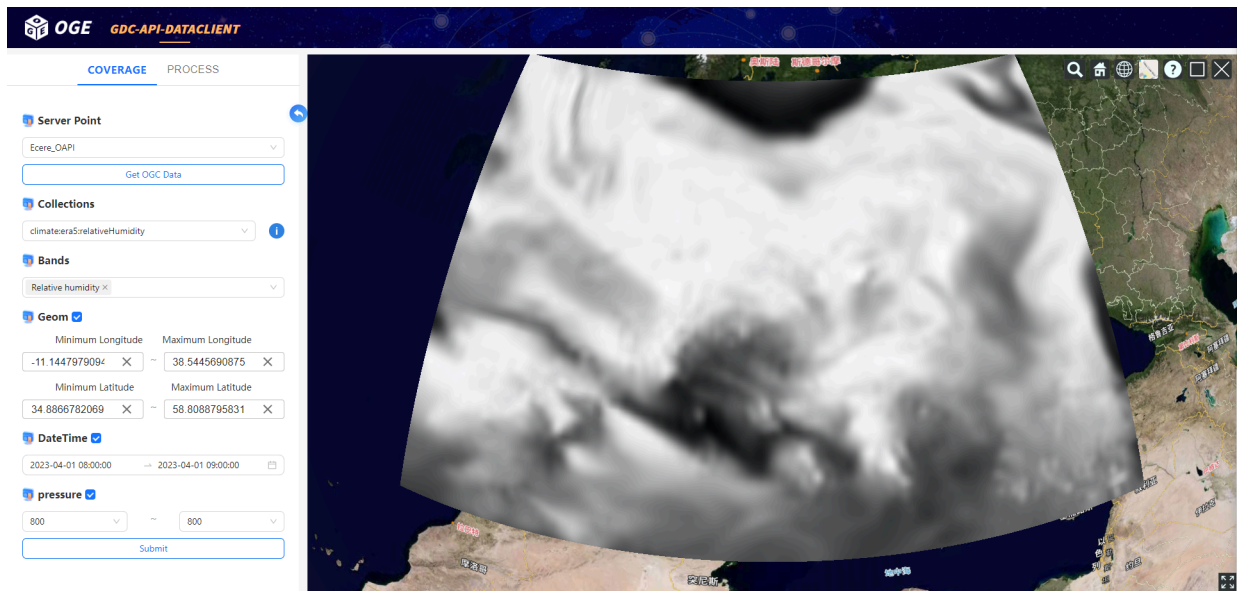


Figure 23 – WHU’s OGE-DataClient visualizing climate:era5:relativeHumidity from Ecere Coverages API

OGC API – Processes

The OGE-DataClient allows users to create execution requests in a JSON editor, including the request body for individual process execution and nested workflows. Supported execution modes include synchronous, asynchronous, and collection output. In collection output mode, users can easily reuse the same workflow by applying different conditional retrievals to the returned Coverage.

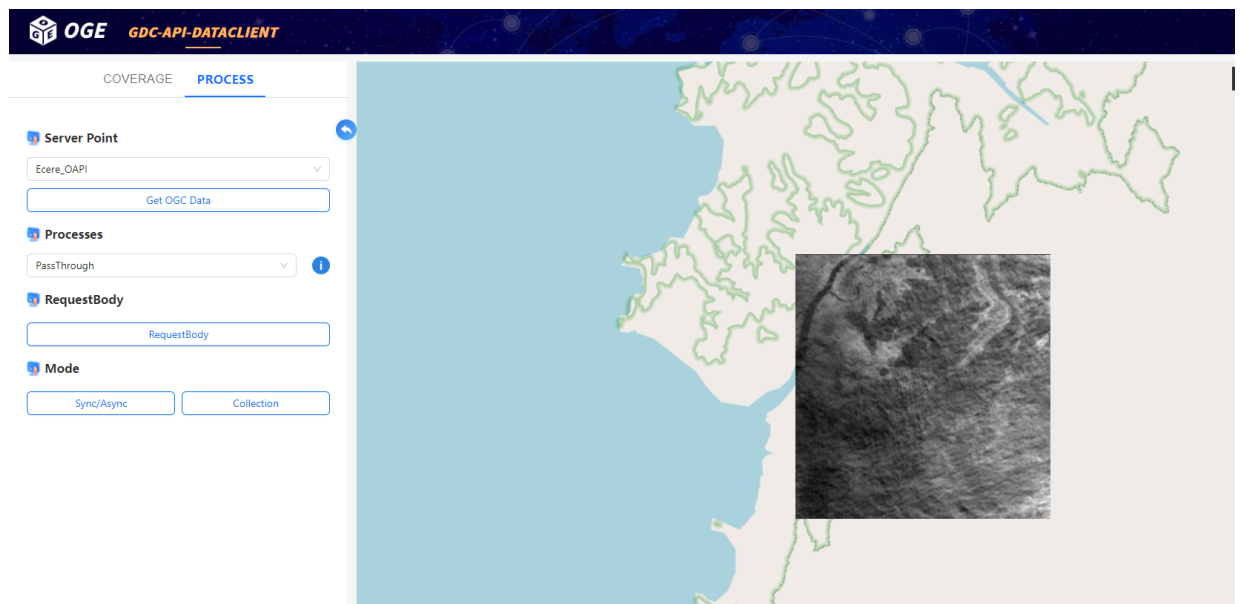


Figure 24 – WHU’s OGE-DataClient visualizing processing result from Ecere Processes API using synchronous mode

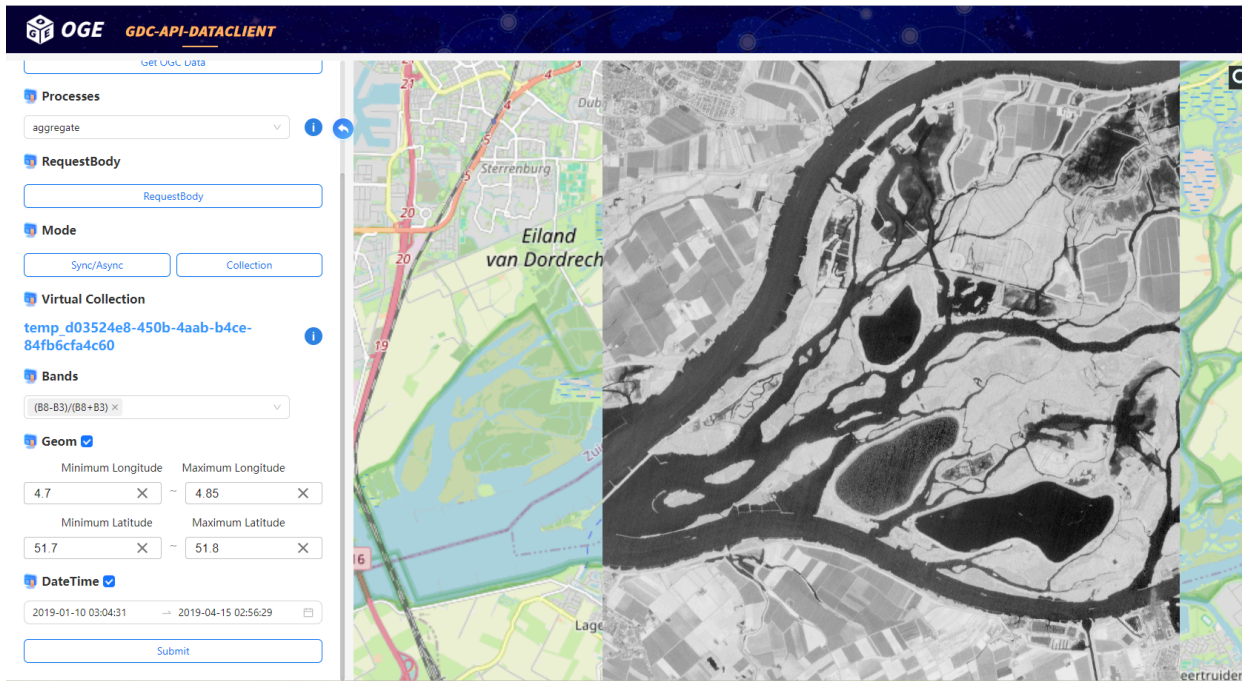


Figure 25 – WHU’s OGE-DataClient visualizing processing result from WHU Processes API using Workflows&Chaining(collection output)

OpenEO API

The OGE-DataClient supports invoking processes provided by the OpenEO API in synchronous mode by creating a process graph in the JSON editor.

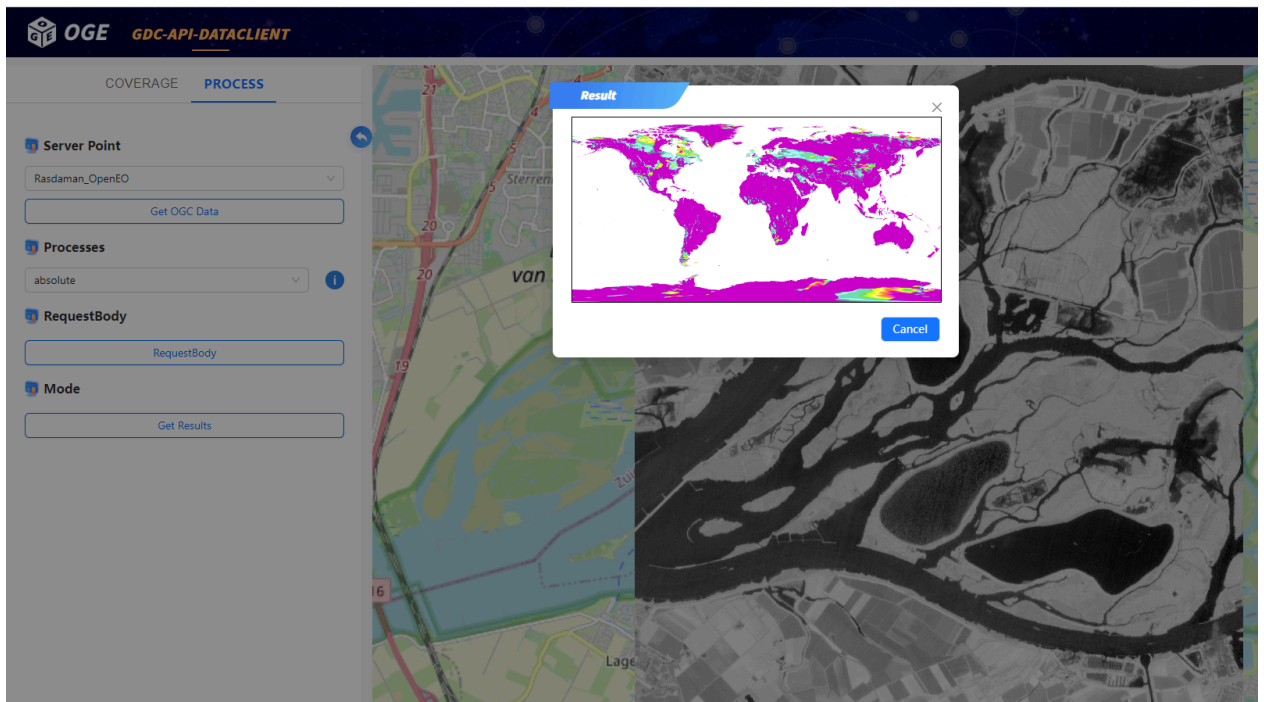


Figure 26 – WHU’s OGE-DataClient visualizing processing result in PNG format obtained from the rasdaman OpenEO API while operating in synchronous mode

5.2.5.3. Future improvements

- Support for the OGC API – Tiles and the draft OGC API – Maps Standards.
- Support for the scaling functionality defined in the OGC API – Coverages Standard.
- Automatically generate a visual interface for input parameters based on the description of the process, simplifying the process of creating the request body.
- Consider rendering and map visualization solutions for large-scale imagery data, avoiding an increase in computational load on the resource-limited browser side.
- Some endpoints of the Coverage API use coordinate systems other than EPSG:4326 for spatial filtering. It is necessary to implement coordinate system conversion for the spatial range selected by the user to support the normal operation of subset operations.

5.2.6. Pelagis Data Solutions – Data Client (D113)

The concept of a GeoDataCube for analysis over both space and time is a very powerful mechanism. The inherent capabilities and proposed standardization present advanced users with a framework to develop complex spatial and temporal analysis both for environmental monitoring and machine learning use cases. It is important that the approach taken fits within

existing OGC Standards baselines and, in a best case scenario, provides these features packaged in a standard's compliant framework for use within existing geospatial applications.

The Pelagis D113 client is designed to exploit the capabilities of the GeoDatacube working products with application to climate monitoring of the Canadian Arctic. The client framework is designed to complement the work efforts associated with the OGC Federated Marine Digital Arctic initiative and demonstrates the integration of the GeoDataCube as a provider of essential climate indicators compliant with the Environmental Data Retrieval (EDR) API.

Background

Pelagis is an OceanTech venture located in Nova Scotia, Canada. The foundation focuses on the application of open geospatial technology and standards designed to promote the sustainable use of our ocean resources. As a member of the Open Geospatial Consortium, Pelagis acts as co-chair in the OGC Marine Domain Working Group (MarineDWG) responsible for developing a spatially-aware federated service model for marine and coastal ecosystems. The immediate priority is the development of a federated marine reference model based on the core OGC standards in support of Nature-based Climate Resilience & Adaptation initiatives.

5.2.6.1. Approach

Satellite Derived Observations for Sea Ice Albedo

A key essential variable for the Arctic is surface albedo which is a strong indicator of a warming Arctic region. The Arctic surface air temperature is rising twice as fast as the rest of the world, and Arctic sea ice is retreating up to three times faster than the rate projected by climate model simulations. The decline of sea ice and snow cover in the Arctic Ocean leads to a decrease of the surface reflection and an increase of absorption of solar radiation which in turn increases surface temperature and triggers sea ice decline. This surface albedo feedback loop is one of the major drivers of Arctic amplification and a strong indicator of a region in transition towards an ice-free environment.

Arctic Regional Reanalysis datasets at 2.5 km resolution are ingested into a rasdaman GDC instance. The dataset covers a circumpolar region including Greenland, the Labrador Sea, and the Davis Strait. This provides a baseline of spatial and temporal characteristics of the region and over which the GDC client performs regional analysis of the surface albedo for an area of interest surrounding the Ninginganiq National Wildlife Area.

As apparent, the change in albedo between the period of May 2020 and April 2023 is substantial in the southern extent while the changes specific to the Ninginganiq National Wildlife Area have remained relatively stable.

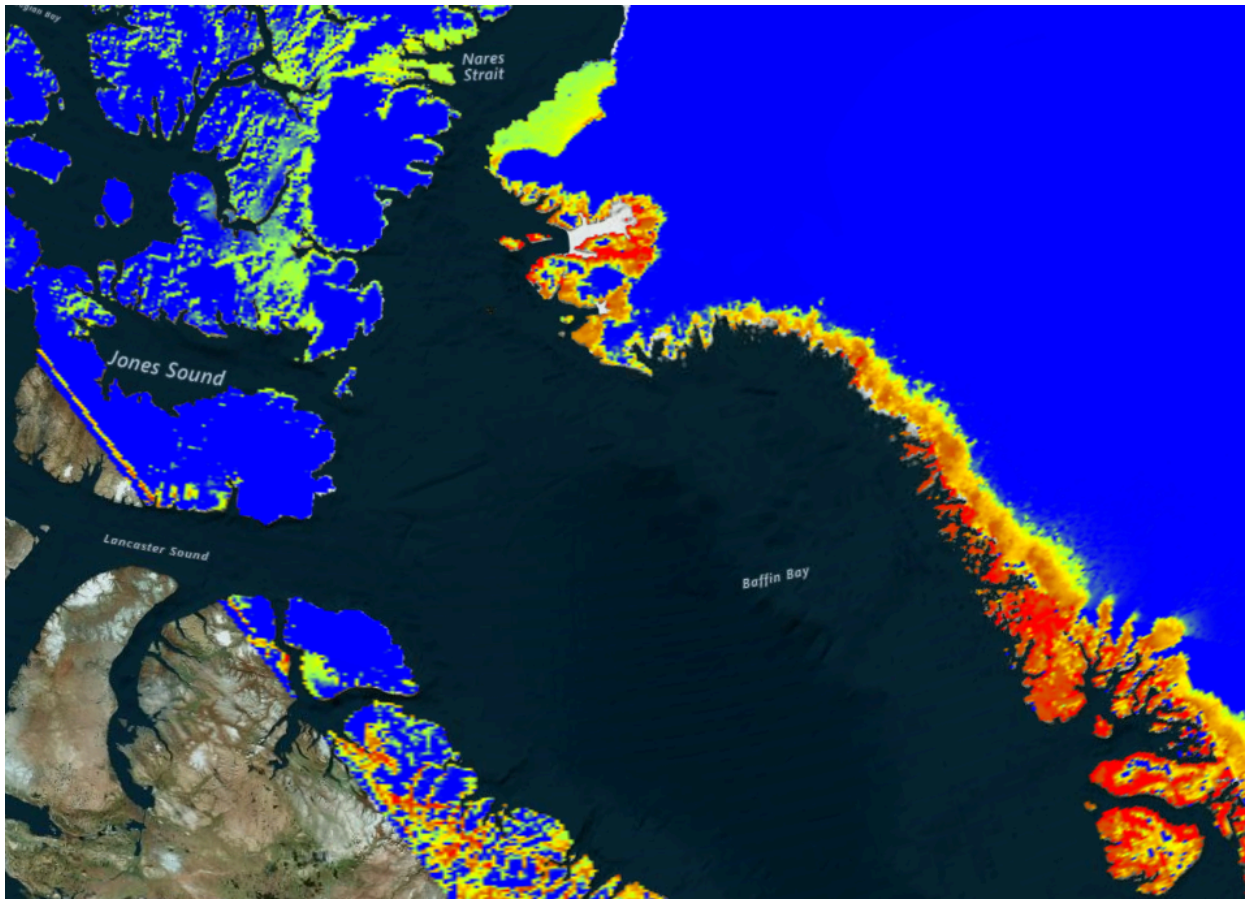


Figure 27 – Albedo as an Essential Variable of the Arctic

Environmental Data Retrieval

The D113 data client is based on the pygeoapi open source platform. The D113 data client is designed as a provider extension that transforms standard Environmental Data Retrieval (EDR) API structured queries to the GDC interface and dispatches the queries to the GDC service provider. Effectively, the GDC provider instance acts as a facade to the GDC API endpoint that provides similar capabilities including slicing, filtering, and trimming across a multi-dimensional cube.

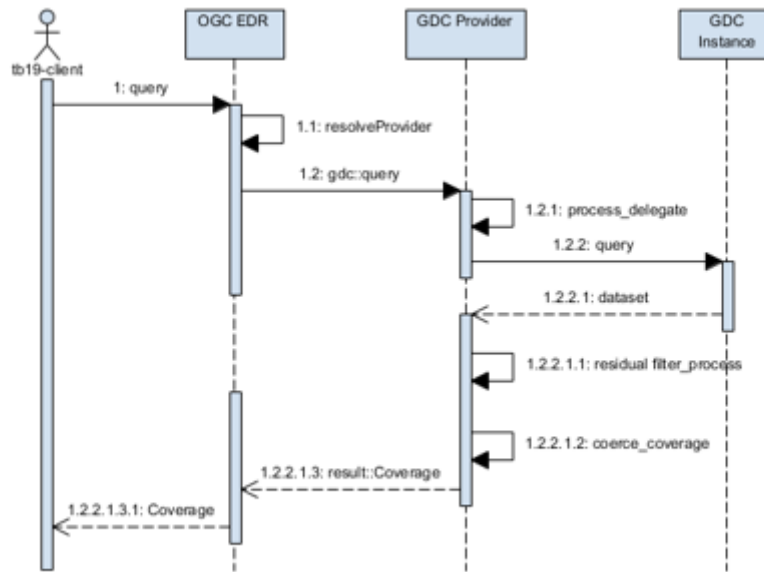


Figure 28 – EDR provider framework

5.2.6.2. Use Cases

Surface Albedo for the Spring Equinox 2019

This use case focuses on using a GeoDataCube API implementation to request slicing the surface albedo observations along the spatial dimensions for the region of interest at a specific time instance. The client requests the 'albedo' parameter from the GDC service provider given an area of interest (polygon) and time instance (datetime=2019-05-06T00:00:00.000Z).

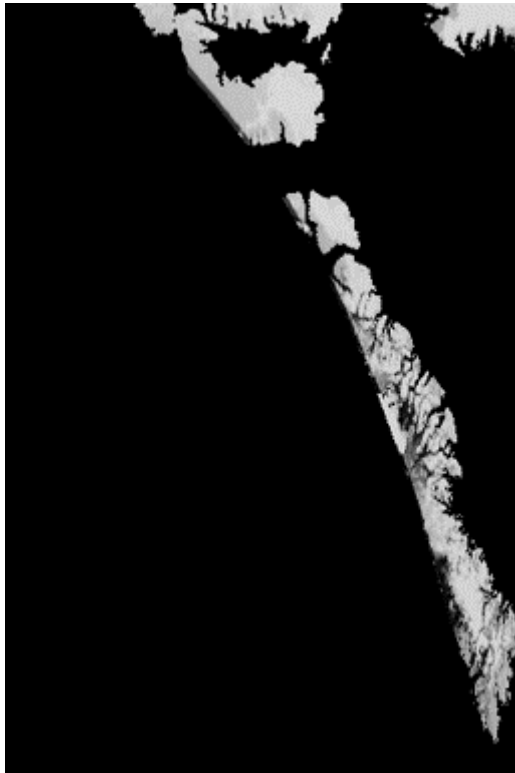


Figure 29 – Surface albedo Spring 2019

Of note is that by implementing the EDR GDC provider as a facade to each GDC service provider, the spatial and temporal constraints may be issued to a separate GDC service provider with no impact to the client workflow. As an example, the following query requests the near-surface temperature from the Ecere GDC provider instance.

Example: /collections/tb19/arctic/instance/cmip5/area?coords=POLYGON -109 64.39894807, -61.25 64.39894807, -61.25 76.7652818, -109 76.7652818, -109 64.39894807&datetime=2019-05-06T00:00:00.000Z¶meter_names=tas&crs=EPSG:4326'

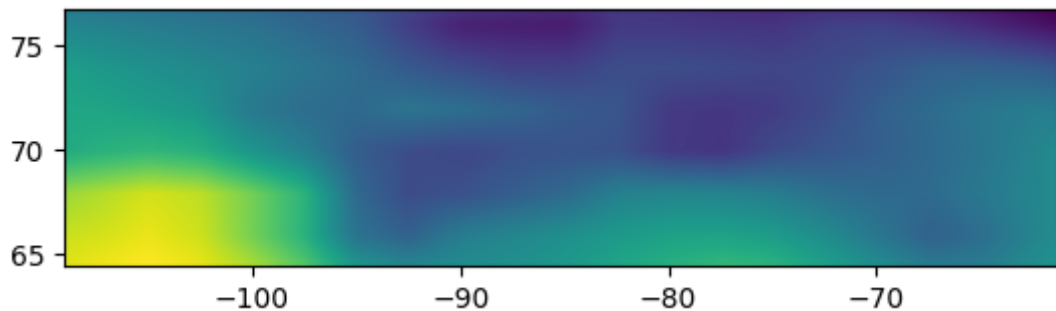


Figure 30 – CMIP5 Near Surface Temperature Spring 2019

Temporal analysis of Surface Albedo

The inherent value of the GeoDataCube framework is its ability to scale both in terms of volume of data and processing capabilities. This use case focuses on delegating the analysis of surface

albedo to the GDC service provider over a temporal extent to determine the magnitude of change for each gridded observation. In this case, the GDC provider translates the EDR request to compose a process graph for execution by the GDC service instance to determine the change in surface albedo for the region of interest over the temporal range of May 2019 – April 2023.

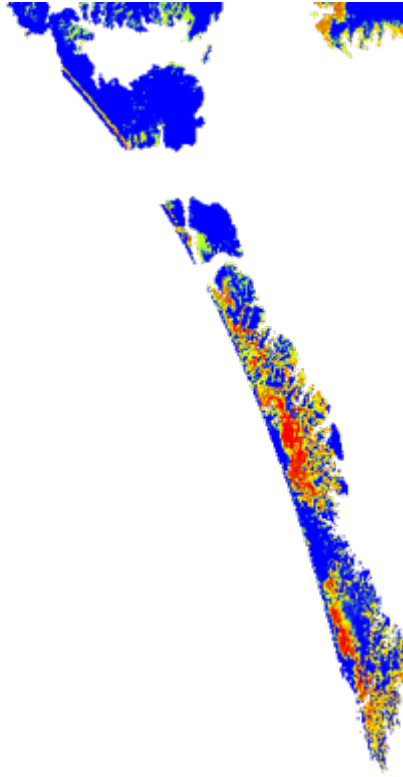


Figure 31 – Magnitude of change in surface albedo between May 2019 and April 2023

5.2.6.3. Challenges & Future Work

EDR API

EDR, in its role as a simplified API for querying discrete coverages of observations, provides an effective means to query across the spatial and temporal dimensions of a GeoDataCube. Certain capabilities of the draft GeoDataCube API standard are abstracted away as EDR queries against a collection/instanceID where the instanceID represents a named process graph. Whether this is the intention of the OGC EDR API standard is to be further investigated as certain process graphs, especially resource intensive process graphs, require an asynchronous workflow that benefits from caching at the service layer.

The EDR query capabilities were limited to investigating area-based and cube-based queries against the GDC service instances. Further investigation into the capabilities of the GeoDataCube framework for trajectory-based and corridor-based queries, for example, would be of benefit to support further interoperability requirements with the OGC Moving Features Standard.

Response Encodings

The response encodings supported by the EDR client focused on image-based (.png), coverage-based (CoverageJSON), and multi-dimensional encodings supported by NetCDF. Further investigation into alternative file-based encodings such as Zarr and GeoParquet would likely enhance the workflow with support for cloud-optimized analysis ready data stores.

5.2.7. 52°North GmbH – Executable Test Suite (ETS) for the draft OGC API – GDC Standard

The Executable Test Suite (ETS) for the draft OGC API – GDC Standard was created to conduct systematic tests of the GDC implementations. The ETS is based on the OGC Team Engine and was implemented using the TestNG Java framework.

The following conformance classes were implemented from scratch.

- Capabilities
- Account Management
- Data Discovery/Access
- Process Discovery
- OpenEO

The following existing Executable Test Suites were included by reference.

- OGC API – Processes Core
- OGC API – Coverages Core
- OGC API – Features Core

The implementation of the tests was straight forward using the detailed description of the functionality of the draft OGC API – GDC Standard. Due to a current lack of schemas there is no validation in place for responses. Instead, the existence of certain key elements is checked. The existing test suite is also based on the TestNG framework, which made reusing the tests easy. However, as not all implementations of the OGC API – GDC are supporting all functionality, a mechanism was implemented to check for conformance first, before calling a referenced ETS. The details can be found here: <https://github.com/52North/ets-ogcapi-gdc10/blob/main/src/main/java/org/opengis/cite/ogcapigdc10/TestListener.java>

- The source code of the ETS for draft OGC API – GDC Standard is available at <https://github.com/52North/ets-ogcapi-gdc10>.
- The hosted demo version of the ETS for draft OGC API – GDC Standard is available at <https://19.testbed.dev.52north.org/teamengine/>.

5.2.7.1. Implementation details

This section lists the tests that the ETS currently consists of.

5.2.7.1.1. Capabilities

The following tests were implemented.

- test Landingpage
- test Http
- test Validate Conformance Operation And Response

5.2.7.1.2. Account Management

The following tests were implemented.

- test Basic Auth
- test User Information

5.2.7.1.3. Data Discovery/Access

The following tests were implemented.

- test Basic Metadata
- test Retrieve Coverage Domainset
- test Retrieve Coverage
- test Retrieve Coverage Rangetype
- test Full Metadata
- test Metadata Filters

5.2.7.1.4. Process Discovery

The following tests were implemented.

- test Process Graphs

- test Process List

5.2.7.1.5. OpenEO

The following tests were implemented.

- test List Batch Jobs
- execute Sync
- test Metadata Of Batch Job
- test Get File Formats

5.2.7.1.6. OGC API – Processes

All tests of core conformance class were included.

5.2.7.1.7. Coverages Core

All tests of core conformance class were included.

5.2.7.1.8. Features Core

All tests of core conformance class were included.



6

INTER COMPARISON EXPERIMENTS

6.1. Ecere Technology Integration Experiments

Ecere performed Technology Integration Experiments with the implementations available from all other participants.

Table 11 – Status of the GNOSIS Cartographer and/or `gdc-test` client integration with participant API endpoints

CAPABILITY / SERVER	ECERE	RASDAMAN	WHU	COMPUSULT	GEOLABS	BROCKMANN	EURAC
<u>Common-1</u> (Core, Landing Page)	S	S	S (server often unavailable)	S	S	S	S
<u>Common-2</u> (Collections)	S	S	S	S	S	S	S
<u>Coverages-1</u> (Core)	S	S	S (504 gateway timeout issues)	⚠ (no overlap in supported formats)	∅	S	S (difficulties: NaN, NODATA, invalid/undeclared mixed UTM zones native CRS)
<u>Coverages-1</u> (Subsetting)	S	S	S	∅	∅	S	S (missing subset parameter, 204 or 500 for some slicing/trimming)
<u>Coverages-1</u> (Scaling)	S	S	∅ (implemented functionality removed for stability)	∅	∅	S	∅

CAPABILITY / SERVER	ECERE	RASDAMAN	WHU	COMPUSULT	GEOLABS	BROCKMANN	EURAC
Coverages- 1 (Field Selection)	S	S	S	○	∅	S	S (missing range type / schema)
Coverages- 1 (Coverage Tiles)	S	∅	∅	∅	∅	∅	∅
Coverages- 2 (CQL2 Filtering)	S	∅	∅	∅	∅	∅	∅
Coverages- 2 (CQL2 Derived Fields)	S	∅	∅	∅	∅	∅	∅
Processes-1 (Core, OGC Pr.Desc.)	S	∅	S	S	S	∅	∅
Processes- 3 (Collection Input)	S	∅	∅ (should be easy to implement based on <i>load Cube</i> process)	∅	∅	∅	∅
Processes- 3 (Collection Output)	S	∅	S	F	∅	∅	∅
Processes- 3 (Nested Processes)	S	∅	S	∅	⚠ (issue with default raw response, conformance decl.)	∅	∅

S = Tested and functional

F = Tested and does not work

○ = Not tested

∅ = Does not exist/Not implemented

🔧 = In development

⚠ = Error / Problem

6.1.1. Experiment examples

This demonstrates visualizing a Tree Cover Density data cube available from the rasdaman GDC API implementation with the client.



Figure 32 – Tree Cover Density dataset from rasdaman's GDC API visualized in 3D using GNOSIS Cartographer

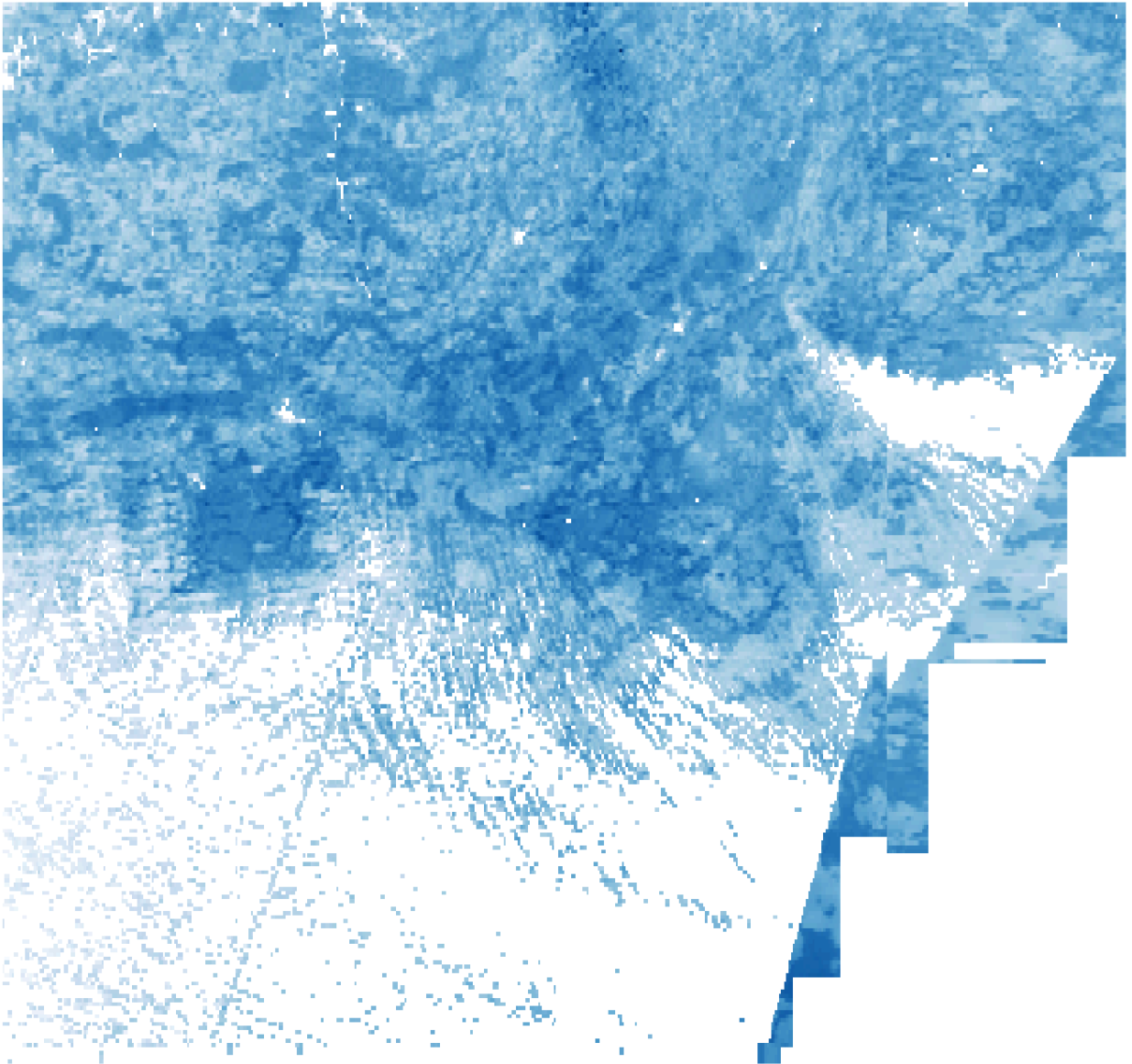


Figure 33 – EVAPOTRANSPIRATION dataset from Brockmann Consult retrieved from Ecere’s gdc-test client (styled in QGIS)

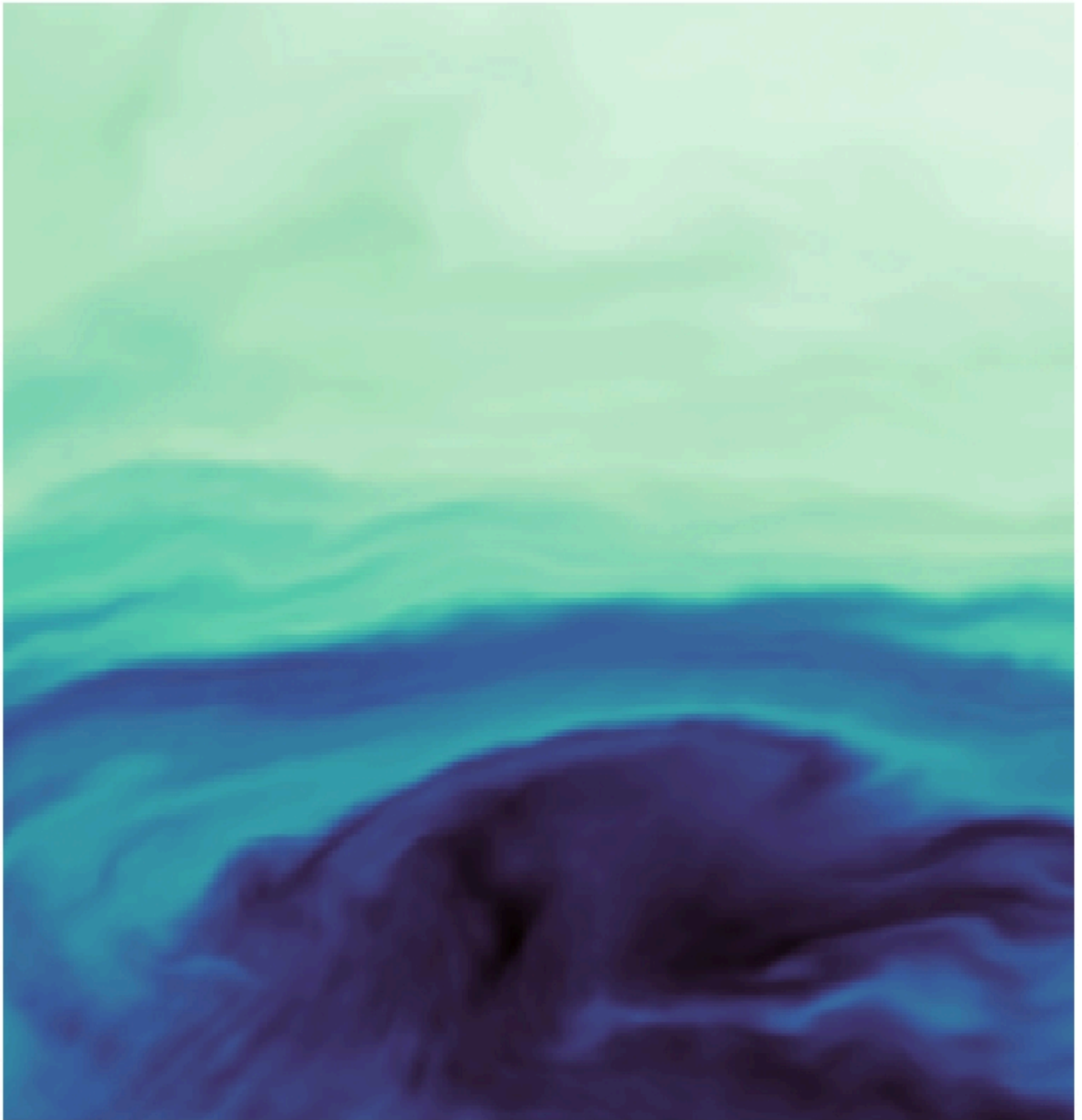


Figure 34 – Successful coverage request from Wuhan University server ECMWF_hsvs collection (styled in QGIS)



Figure 35 – Successful coverage request from Eurac server s2_I2a collection

To provide an easier and objective way to continuously test different server implementations and report TIE success, Ecere developed a command-line GDC API test application. The test reports a PASSED, FAILED, or SKIPPED result for each of the supported GDC API capabilities.

The tool supports several options to select testing with a specific collection, temporal interval, spatial region, or fields. The test considered both retrieving data as raw coverage as well as attempting to render it by applying a style, and generate an output which can be kept for further manual validation. A pre-defined execution request must be provided to test the Processes capabilities, as the tool will not yet attempt to generate one automatically from a process.

This figure demonstrates the execution of the GDC test tool using the Ecere GNOSIS Map Server API endpoint.

OGC Testbed 19 - GeoDataCube API Test Suite, by Ecere Corporation
 Syntax:

```
obj/release.linux/gdc-test [[test | keep | prepare | clean] <http(s) endpoint> [<outputs dir>]] [options]
```

where option can be any of:

```
-credentials      <user>:<password>
-bearerToken     <bearer token>
-collection      <selected {collectionId}>
-subset         <subset string>
-time           <RFC3339 date/time>
-scale          <scale factor (2: twice downsampld)>
-crs            <CRS URI or CURIE>
-format         <extension>
-fields         <selected or derived fields[,...]>
-tms           <TileMatrixSet>
-tile          <level,row,column>
-style         <style sheet to use (SLD/SE, MapboxGL, or GNOSIS CMSS)>
-execRequest1   <JSON execution request file for Processes-1 Core Execution>
-execRequest2   <JSON execution request file for Processes-3 (Collection Output)>
-process        <selected {processId}>
```

Test preparation failed for GDCAPITest.

Some tests or preparation FAILED.

Figure 36 – Ecere’s GDC Test tool showing syntax help for command line arguments

```
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage?f=tifs&scale=size&lon(259),Lat(259)&properties=Elevation%20%20106&filter=Elevation%20%3E%200.5&subset=Lat(-0.351562500000-90.3515625000000),Lon(-0.3515625000000-90.3515625000000)
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage?f=tifs&scale=size&lon(259),Lat(259)&properties=Elevation%20%20106&filter=Elevation%20%3E%200.5&subset=Lat(-0.351562500000-90.3515625000000),Lon(89.6484375000000-180.3515625000000)
Rendering screenshot, it might take a while...
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/0.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/1.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/2.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/3.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/0.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/1.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/2.gmt
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/3.gmt
Rendering screenshot, it might take a while...
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/0.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/1.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/2.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/3.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/0.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/1.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/2.gmt?filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/3.gmt?filter=Elevation%20%3E%200.5
Rendering screenshot, it might take a while...
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/0.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/1.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/2.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/3.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/0.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/1.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/2.gmt?properties=Elevation%20%2010
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/3.gmt?properties=Elevation%20%2010
Rendering screenshot, it might take a while...
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/0.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/1.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/2.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/1/3.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/0.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/1.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/2.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Requesting: https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama/coverage/tiles/GNOSISGlobalGrid/0/0/3.gmt?properties=Elevation%20%20106&filter=Elevation%20%3E%200.5
Rendering screenshot, it might take a while...
**PASSED** [GDCAPITest:Coverages-1 (Core)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-1 (Subsetting)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-1 (Scaling)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-1 (Field Selection)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-1 (Coverage Tiles)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-2 (SQL Filtering)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
**PASSED** [GDCAPITest:Coverages-2 (SQL Derived Fields)]:https://maps.gnosis.earth/ogcapi/collections/SRTM_ViewFinderPanorama] successfully.
Requesting: https://maps.gnosis.earth/ogcapi
Requesting: https://maps.gnosis.earth/ogcapi/processes/AddAttributes
(SKIPPED) [GDCAPITest:Processes-1 (Core, OGC Pr.Desc.)]:https://maps.gnosis.earth/ogcapi] because of no JSON execution request for Sync/Async process execution provided (-execRequest1).
(SKIPPED) [GDCAPITest:Processes-3 (Collection Output)]:https://maps.gnosis.earth/ogcapi] because of no JSON execution request for Collection Output process execution provided (-execRequest3).
Cleaning test outputs (re-run with 'keep' to keep them).
All tests PASSED successfully.
```

Figure 37 – Ecere’s GDC Test tool executed for the GNOSIS Map Server endpoint

6.2. Eurac Research – GDC Web Editor

The following table provides an overview of the current status (February 2024) of the GDC Web Editor connecting to the GDC API implementations.

All the functionality mentioned in the table header is available in the GDC Web Editor except for some specific requirements for workflows in OGC API – Processes – Part 3. If a cell is set to 'n/a' it means that the server doesn't support the functionality.

The Wuhan Server can only be tested with a local dev version of the GDC Web Editor as it is not accessible via HTTPS. The hosted online version of the GDC Web Editor can only access servers supporting HTTPS due to security reasons.

Table 12

API	CONNECTION	AUTH	DATA DISCOVERY	PROCES: DISCOVER	COVERAGE DOWNLOAD	STAC ITEM DOWNLOAD	OGC API PROCESSING	OGC API JOE	OPENEO PROCESSING
Ecere	✓	n/a	✓	✓	✓	n/a	✓ Sync	n/a	n/a
rasdaman	✓	✓	✓	✓ ⚠ derives from openeo-processes:	✓	n/a	n/a	n/a	✓ Sync + UDP
WHU	✓ ⚠ HTTP only	n/a	✓	✓	✓	✓	✓ Async	✓	n/a
GeoLabs	✓	✓	✓	✓	n/a	✓	✓	✓	n/a
Brockmann	✓	n/a	✓	n/a	✓	✓	n/a	n/a	n/a
EURAC	✓	✓	✓	✓	✓	n/a	n/a	n/a	✓ Sync + Async

6.3. Geomatys – API endpoints integration

Below is the status of API endpoint integrations in the Geomatys Unreal Engine (3D) client. These API endpoints are provided by Testbed-19 GDC participants' servers.

Table 13

OPERATIONS / SERVERS	ECERE	RASDAMAN	WHU	COMPUSULT	GEOLABS	BROCKMANN	EURAC
Authentication	-	✓	-	○	-	-	✓

OPERATIONS / SERVERS	ECERE	RASDAMAN	WHU	COMPUSULT	GEOLABS	BROCKMANN	EURAC
GET /collections	✓	✓	✓	○	✓	✓	✓
GET /collections/{cid}	✓	✓	✓	○	○	✓	✓
GET /collections/{cid}/coverage (only geotiff)	✓	○	✓	○	○	○	○
GET /collections/{cid}/coverage/ rangetype	✓	✓	✓	○	○	✓	○
GET /collections/{cid}/coverage/ domainset	✓	✓	✓	○	○	✓	○
GET /collections/{cid}/coverage/ tiles (only png/jpg)	✓	○	○	○	○	○	○
GET /collections/{cid}/map	○	○	○	○	○	○	○
GET /collections/{cid}/map/tiles	✓	○	○	○	○	○	○

6.3.1. Legend

- ✓ = Tested and functional
- ✗ = Tested and does not work
- = Not tested
- = Does not exist / Not implemented
- ⚠ = Error / Problem

6.3.2. Geomatys Client Supported Capabilities

Geomatys supports OGC API endpoints (not OpenEO or STAC API).

- OGC API – Coverages (subsetting, scaling, field selection) ✓ (only with png/jpg/geotif and not tiled)
- OGC API – Maps ✓ (only tiled rasters)

Some documentation of APIs supported by our client :

- [Common-1 \(Core,Landing Page\)](#)
- [Common-2 \(Collections\)](#)

- [Coverages-1 \(Core\)](#)
- [Coverages-1 \(Subsetting\)](#)
- [Coverages-1 \(Scaling\)](#)
- [Coverages-1 \(Field Selection\)](#)
- [Maps-1 \(Core\)](#)

6.4. Wuhan University Technology Integration Experiments

This table illustrates the interaction testing of each GDC API endpoint using the data client implemented by Wuhan University.

Table 14

API	GET COLLECTION	GET COLLECTION	GET DOMAINSE	GET RANGETYP	GET COVERAG	GET PROCESSE	GET PROCES	EXECU OAP -1	EXECU OAP -3	EX OAP (S)
Ecere	✓	✓	✓	✓	✓	✓	✓	✓ Sync	✓	N/A
rasdaman	✓	✓	✓	✓	✓	✓	N/A	N/A	N/A	✓
WHU	✓	✓	✓	✓	✓	✓	✓	✓+ Async	✓	N/A
GeoLabs	✓	✓	N/A	N/A	N/A	✓	✓	✓	N/A	N/A
Brockmann	✓	✓	✓	✓	✓	N/A	N/A	N/A	N/A	N/A
EURAC	✓	✓	N/A	N/A	N/A	✓	N/A	N/A	N/A	○
Compusult	○	○	○	○	○	○	○	○	○	○

✓ = Functional

✗ = Does not work

○ = Not tested

N/A = Not implemented

6.5. 52°North GmbH – Executable Test Suite (ETS) for the draft OGC API – GDC Standard

The following table provides an overview of the current status (February 2024) of the GDC API implementations that were tested using the ETS for OGC API – GDC.

The list of tested functions can be found below. The number in brackets indicates passed/failed/skipped tests.

Table 15

PROVIDER	CAPABILITIES	ACCOUNT MANAGEMENT	DATA DISCOVERY ACCESS	COVERAGES CORE	FEATURES CORE	PROCESS DISCOVERY	OGC API PROCESSES	OPENEO
Ecere	2/1/0	0/1/1	4/2/0	passed	230/1/47	1/1/0	passed	1/2/1
rasdaman	2/1/0	passed	3/3/0	skipped	2/3/12	passed	skipped	2/1/1
WHU	2/1/0	0/1/1	passed	skipped	5/3/2	1/1/0	17/19/0	1/2/1
GeoLabs	2/1/0	0/1/1	3/3/0	skipped	2/3/12	1/1/0	passed	1/2/1
Brockmann	passed	0/1/1	3/3/0	skipped	4/3/10	0/2/0	skipped	1/2/1
EURAC	passed	passed	2/4/0	skipped	skipped	1/1/0	skipped	passed

6.5.1. Known limitations

The referenced test suites (ETS API – Processes, Coverages, and Features) need an API description in order to work correctly due to the prototypical nature of the tested implementations. As such, this API description was not always available. In this case, the respective referenced tests were skipped.

7

USABILITY TESTS

7.1. Sinergise Usability Test

To perform usability tests of the draft GDC APIs and clients, the decision was made to replicate an existing processing chain that is a part of the agricultural area monitoring system that several EU countries use.

7.1.1. Test Scenario Definition

Inputs:

- GeoPackage with 100,000 field polygons, each having a “crop type group” assigned; and
- Sentinel-2 L2A data for 2022 (all observations).

Desired result:

- An application that compares Normalized Difference Vegetation Index (NDVI) timeseries of a polygon with its neighbors - <https://area-monitoring.sinergise.com/docs/markers/similarity-and-euclidian-distance-marker/examples/#similarity-marker-examples> (see the application screenshots).

Implementation Steps:

- for each of the input polygons:
 - calculate NDVI of pixels within the polygon;
 - compute statistics over all pixels belonging to the same polygon – min/max/mean/percentiles; and
 - repeat for each observation in the time interval (full year);
- calculate similarity scores by the back-end using the formula defined here: <https://area-monitoring.sinergise.com/docs/markers/similarity-and-euclidian-distance-marker/#similarity-score>;
- configure UI widget showing the NDVI time series of the selected object compared with similar objects in the neighborhood; and

- configure UI widget showing time-lapse of the satellite images of the selected parcel using one of various visualizations (true color, false color, NDVI).

After initial analysis of the capabilities of the proposed draft GeoDataCube (GDC) API Standard, available servers, and clients, the Testbed 19 participants decided that the execution of all the implementation steps (above) was too ambitious for completion in Testbed 19. Therefore, the decision was to use a more streamlined scenario, removing the requirement for calculation of similarity scores and configuration of UI widgets.

7.1.2. Principles of Usability Testing

In usability testing, the selected scenario was executed, including discovery, implementation, and execution phases that are typical in development of an EO processing chain.

Since the purpose of the Testbed 19 GeoDataCube (GDC) thread was the evaluation of a proposed OGC GDC API Standard, a typical user interface usability test was not done. Instead, the primary focus was on the usability aspects of the GDC API and the clients' support of the draft GDC API, such as:

- evaluate conceptual alignment of the proposed GDC API for the tested scenario;
- identify any gaps in the ability of the draft GDC API Standard to fully support the scenario implementation;
- evaluate ease-of-use in implementations of the GDC API, as presented via the client in discovery, implementation, and execution phases of the scenario;
- identify any missing support of required GDC API functionality in the implementation of the client; and
- identify opportunities in clients for improvements of the user experience over the basic functions of the GDC API.

7.1.3. Usability Testing Results

Implementations of the proposed GDC API were tested using the Eurac GDC Web Editor Client and, in parallel, trying to use GDC API calls directly from Python to be able to better understand the behavior of the UI client.

The general conclusion of the executed scenarios is that the draft GDC API Standard is suitable for implementation of the use-case. Most of the difficulties were caused by the fact that the server and client UI implementations of the draft GDC API are not fully mature or do not support all the capabilities envisaged for the GDC API Standard.

The few desired functionalities that are currently not supported by the draft OGC GDC API Standard would not be too difficult to work around using client libraries and a small amount of data post-processing.

Table 16

SCENARIO ACTIVITY	USABILITY ASSESSMENT
discover coverage dataset	<ul style="list-style-type: none"> ✓ API provides ample metadata about available coverages. i With large lists of coverages, discovery can be slow: The draft GDC API does not mandate paging. i Filtering the returned coverage is not supported by the draft GDC API. ✓ The GDC Web Editor client provides neatly formatted metadata and search capability.
visualize coverage	<ul style="list-style-type: none"> ✓ Downloading a subset of a coverage is possible through the coverage endpoint. ✎ The GDC Web Editor client does not provide an easy way to 'preview' the data. The user must create a new process or use the coverage download wizard. ✓ Most UI clients do a reasonable job at default rendering of Sentinel 2 data and NDVI.
list available timestamps	<ul style="list-style-type: none"> ✓ The OGC API – Coverages Standard specifies requirements for retrieving the DomainSet of a coverage. ✓ STAC provides Temporal Extent info that is neatly rendered in the GDC Web Editor client. ✎ It would be helpful to have some position-specific filtering of timestamps (available observations over some AOI), lest the user is left to trying several dates to find one that has the desired coverage data.
implement band selection	<ul style="list-style-type: none"> ✓ The draft OGC API – Coverages Standard specifies requirements for subsetting bands when retrieving coverages. ✓ The openEO load_collection process has a convenient bands parameter filter. ✓ The GDC Web Editor client supports drop-down selection from a preset list of collection's bands. i Sentinel-2 coverages in the rasdaman server are single-band, while the EURAC server combines all bands into a single coverage. The recommendations for encouraging consistency between server implementations would be beneficial for the user experience.
visualize selected bands	<ul style="list-style-type: none"> ✓ Support for visualizing rasters with arbitrary bands in the GDC API is basic, but reasonable. ✎ openEO requires a save_result process at the end, which is inconvenient when just doing data exploration.
discover band math capabilities	<ul style="list-style-type: none"> ✓ The openEO API defines processes and allows listing those supported by a server. ✓ The GDC Web Editor client provides free-text search capabilities over processes and their metadata. ⚠ The OGC API – Processes Standard does not prescribe 'basic' processes. The availability of basic math is up to the server.
implement band math	<ul style="list-style-type: none"> ✓ The GDC Web Editor client provides an excellent wizard to convert mathematical expressions into process graphs. ✎ The openEO API does not broadcast mathematical operations over data cubes (e.g., numpy https://numpy.org/doc/1.26/user/basics.broadcasting.html). This takes some time for the user to adjust to. ✎ The openEO API broadcasting is implemented differently in rasdaman (allows broadcast) and EURAC (does not allow broadcast, requires reduce_dimension process). ✎ The openEO API processes for broadcasting over a data cube are difficult to learn - apply, apply_dimension, and reduce_dimension all seem like good candidates for the task, but finding the correct one is often trial-and-error.

SCENARIO ACTIVITY	USABILITY ASSESSMENT
	<ul style="list-style-type: none"> ✔ Processing over specific dimensions on generic data cubes is not an easy subject. The openEO documentation provides comprehensive documentation and a tutorial on this topic.
clipping to polygon AOI	<ul style="list-style-type: none"> ✔ The openEO API supports clipping the coverage with a polygon in the <code>spatial_extent</code> parameter of the <code>load_collection</code> process. ⚠ The OGC API – Coverages Standard requirements for retrieval support axis-parallel bounding-box subsetting only. ✎ Support for polygon clipping in <code>load_collection</code> is not common in current openEO servers.
averaging over AOI	<ul style="list-style-type: none"> ✔ The openEO API provides the <code>aggregate_spatial</code> process that supports this requirement well. ✔ The resulting vector data cube can be encoded in JSON. ✎ openEO higher-order functions (aggregations, apply, reduce, etc.) are not easy to develop with. This is mostly due to a critical dependency on the dimensions of the data cubes that are being processed. The recommendation is that this information is made available to developers while designing the process graphs. ✎ openEO process execution errors are often not helpful. Providing more verbose logging and a more structured error handling with an indication of the location of the error in the source code (or graph) would make for a smoother developer experience. (The cause of this problem was likely the prototype state of server implementations and not a shortcoming of the draft GDC API).
apply the process over many AOIs	<ul style="list-style-type: none"> ✔ openEO's batch execution mode supports scaling the process over large areas and many AOIs. ✎ openEO supports parameterized process graphs, but parameterizations are not supported by the GDC Editor client, unless the server supports storing a process graph.
inspect the full results of the processing	<ul style="list-style-type: none"> ⚠ The vector data cube that is the result of the processing is not a coverage available through an implementation of the draft OGC GDC API Standard. The recommendation is adoption of some best practices aimed at harmonizing the use of vector data cubes across all OGC APIs related to processing of Earth observation data.



8

LESSONS LEARNED FROM API IMPLEMENTATION

LESSONS LEARNED FROM API IMPLEMENTATION

Generally, the execution of Testbed 19 was needed at this point in time given the number of available GDC solutions. For the sake of avoiding technological fragmentation and allowing interoperability to happen, considerable exchange and discussion were needed to bring the best of the different solutions “under one roof.” A key aspect to this work was to start with a clear GDC API specification to develop against. Much time was lost in the early phases of Testbed 19 since no GDC API specification was available. In order to come up with a draft specification, a deep understanding of the necessary existing solutions was required.

In order to meet this starting requirement, a series of crosswalks were defined that compare key components and details of different existing standards and specifications that include information on where the standards align and where the standards are conflicting. This was true for both the data discovery part as well as the data processing part. One of the key deliverables of the Testbed 19 GDC activity is the comprehensive crosswalk comparisons of the technologies. This step required the willingness of all participants to take step back and look at other solutions in detail before engaging with development activities of any kind. For future iterations of the OGC GDC work, additional effort into this screening and understanding of the technological landscape should be considered.

The current draft GDC API as developed and presented in Testbed 19 has to be understood in this context as well. The current draft was not intended to be a perfect solution for datacube processing. This is partly due to the fact that the different approaches mainly developed in OGC API processes and openEO are not yet consistently aligned. This was done by design to allow implementers to discover and develop against both ways of processing and the intention was to allow the Testbed usability studies to comment on the different approaches.

At the level of data processing, two fundamentally different models of how to allow for processing on the web are in use. One is based on the concept of wrapping existing code into application packages and running the packages in a supporting execution environment. The other is based on allowing for complete design of workflows based on a set of predefined functions or processes. These two paradigms come with different advantages and drawbacks. The first allows for very easy integration of legacy tools and code. The latter allows for very efficient design of processing workflows especially when considering the chaining of operators and processes into long and complex workflows. However, the developer is required at least once to recode the workflow using the predefined processes. On the other hand, from a computational efficiency point of view it is very hard to optimize workflows not knowing the implementation details of every node or process in the graph. This can lead to performance losses when chaining multiple such packages into a processing pipeline.

Ideally it would be possible to integrate both approaches and have the possibility to chain legacy code with new developments in one workflow. Users should prioritize the use of highly optimized predefined processing functions and rely on application packages only when being absolutely forced to utilize legacy code, e.g., for reproducing results of research. This still requires work both conceptually and on the implementation side. Some ideas contained in the draft OGC API – Processes Part 3 Standard can be used for this. Also an extended more general

definition of user defined functions in openEO allowing to integrate application packages as nodes in the openEO process graph should be considered.

Generally, the use cases presented with Testbed 19 have proven to be challenging given the current state of development and design of the existing OGC API Standard. At the current state of the draft of GDC API, the use cases are all theoretically possible, but more work is needed for the implementations to catch up with the draft specification of the GDC API Standard.

This in turn leads to very little time for running the usability tests, which had to be done in a rushed manner towards the end of the Testbed. The tests did confirm that implementations of the GDC API are capable of tackling complex use cases, but that there still are many issues with the current implementations to support all features of the draft specification. Limitations that have been found regarding the GDC API standard definition include:

1. better support for paging and filtering when servers offer large amounts of collections;
2. data exploration in openEO is non trivial especially when back-ends do not support secondary web services;
3. OGC API Processes does not guarantee support for a set of basic mathematical operators;
4. OGC API Coverages does not allow for polygon based subsetting;
5. results that are not raster based are not easily extracted through OGC API coverages; and
6. currently the user needs to select which way of processing method to follow: The OGC API – Processes or the openEO approach.

On the positive side the participants determined that the draft GDC API Standard is very useful already:

1. access to collection meta data is plentiful and informative;
2. the addition of simple and direct download functionality through OGC API Coverages is appreciated;
3. band filtering, as well as spatial and temporal filtering of collections is well supported;
4. visualizing arbitrary band combinations is reasonable;
5. the guaranteed availability of a well defined set of basic operators is much appreciated; and
6. supporting both synchronous and asynchronous processing is good, to support both prototype development and large scalable processing.

9

FUTURE OUTLOOK

Testbed-19 has continued and furthered an ongoing discussion about how to interact with GeoDataCubes in the most interoperable way. The Testbed participants produced a draft for a specification bringing together the most relevant developments in GeoDataCube technology in and outside of OGC which greatly furthered the common understanding of the available solutions and started a process of discovering to a much better degree the advantages and drawbacks of these solutions. Testbed 19 participants produced prototypes of five back-end implementations and six client implementations as well as an automated test suite. Many of these solutions, being available as open source, providing a perfect starting point for further activities in this direction.

The main issue with the current draft GDC API specification is that the specification still allows for two fundamentally different ways of interaction for describing and triggering processing of data cube resources. The first recommendation would be for the OGC GDC SWG and future OGC Testbeds to consider how these two different approaches can be brought together thereby enabling a flexible yet efficient and integrated interoperable processing workflow design well supported by clients.

The Testbed 19 GDC use cases were challenging, which is good in terms of making sure that the draft specification does not have a narrow scope. This however also brought to light issues in terms of effort required to bring the prototype implementations to the level to be able to run those more complex use cases. Given that the draft GDC API standard was tested against those use cases, a next recommendation would be to continue and extend the testing on usability to get more feedback on the current design of the specification before it goes into the OGC standardization process.

There were also some proposals during the Testbed that the GDC API specification be a profile of existing OGC API building blocks. A clear definition of such profiles is still incomplete and needs more discussion. Also, not all the promising technologies and tools used to create the draft GDC API standard are currently OGC standards and hence cannot easily be referenced in a consistent way. As of February 2024, both the openEO and STAC specifications are proposed as OGC Community Standards, possibly making this interaction more feasible.

As an outcome of the useability tests that were performed, it is also clear that how the datacubes can be connected with other data sources or new types of data cubes based on vectors, for example, needs to be considered.

Apart from how to interact with the GeoDataCubes themselves, there is also a need to harmonize the content of metadata fields and the organization of the content of the data cubes to increase interoperability. This refers to things such as the names of bands in multispectral datasets or the setup of the cubes including all bands in one cube or having a cube for each band. This issue must be discussed certainly also with other tasks and initiatives being pursued in the Analysis Ready Data SWG and related testbed activities as well as possibly CEOS.



A

ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS



ANNEX A (NORMATIVE) ABBREVIATIONS/ACRONYMS

ADES	Application Deployment and Execution Service
AP	Application Package
API	Application Programming Interface
ATLAS	Advanced Topographic Laser Altimeter System
CF	Climate and Forecast (Metadata Conversions)
CIS	Coverage Implementation Schema
CMIP	Coupled Model Intercomparison Project
COPS	Cloud Optimized Point Cloud Specification
CORS	Cross-Origin Resource Sharing
CQL	Common Query Language
DEM	Digital Elevation Model
DSM	Digital Service Model
EDR	Environmental Data Retrieval
EMS	Exploitation Platform Management Service
EVI	Enhanced Vegetation Index
EO	Earth Observation
ER	Engineering Report
GDC	Geo Data Cube
GUI	Graphical User Interface
HTML	HyperText Markup Language

ICESAT-2	Ice, Cloud, and Land Elevation Satellite
ISO	International Organization for Standardization
JSON	Java Script Object Notation
MARS	Meteorological Archival and Retrieval System
MSDI	Marine Spatial Data Infrastructure
NetCDF	Network Common Data Form
NSIDC	National Snow & Ice Data Center
OAC	OGC API Coverages
OAP	OGC API Processes
OAR	OGC API Records
REST	Representational State Transfer
STAC	Spatial Temporal Asset Catalogue
UI	User Interface
WCS	Web Coverage Service
WCPS	Web Coverage Processing Service
WMS	Web Map Service
WMTS	Web Map Tiling Service
WPS	Web Processing Service
XML	Extensible Markup Language



BIBLIOGRAPHY





BIBLIOGRAPHY

- [1] Katharina Schleidt, Ilkka Rinne: OGC 20-082r4, *Topic 20 – Observations, measurements and samples*. Open Geospatial Consortium (2023). <http://www.opengis.net/doc/as/om/3.0>.
- [2] Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r6, *OGC API – Environmental Data Retrieval Standard*. Open Geospatial Consortium (2023). <http://www.opengis.net/doc/IS/ogcapi-edr-1/1.1.0>.
- [3] Emmanuel Devys, Ted Habermann, Chuck Heazel, Roger Lott, Even Rouault: OGC 19-008r4, *OGC GeoTIFF Standard*. Open Geospatial Consortium (2019). <http://www.opengis.net/doc/IS/GeoTIFF/1.1.0>.
- [4] Kyoung-Sook KIM, Nobuhiro ISHIMARU: OGC 19-045r3, *OGC Moving Features Encoding Extension – JSON*. Open Geospatial Consortium (2020). <http://www.opengis.net/doc/IS/mf-json/1.0.0>.
- [5] Douglas Nebert, Uwe Voges, Lorenzo Bigagli: OGC 12-168r6, *OGC® Catalogue Services 3.0 – General Model*. Open Geospatial Consortium (2016). <http://www.opengis.net/doc/IS/cat/3.0.0>.
- [6] *Draft OGC API – Connected Systems – Part 1: Feature Resources* <https://ogcapi.org/connectedsystems/>
- [7] OGC API – Records – Part 1: Core
- [8] OGC Building Blocks: <https://blocks.ogc.org/index.html>
- [9] Lawrence Livermore National Laboratory: NetCDF CF Metadata Conventions: <http://cfconventions.org/>