TESTBED-18: SECURE ASYNCHRONOUS CATALOG ENGINEERING REPORT

ENGINEERING REPORT
PUBLISHED

Submission Date: 2022-11-03
Approval Date: 2022-12-15
Publication Date: 2023-01-05
Editor: Yves Coene, Christophe Noel

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 © 2023 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 ........................................................................................................... xii

II. KEYWORDS .......................................................................................................................... xiii

III. SECURITY CONSIDERATIONS ............................................................................................ xiv

IV. SUBMITTING ORGANIZATIONS ......................................................................................... xv

V. SUBMITTERS ......................................................................................................................... xv

VI. ABSTRACT ........................................................................................................................... xvi

1. SCOPE ................................................................................................................................... 2

2. NORMATIVE REFERENCES .................................................................................................... 5

3. TERMS, DEFINITIONS, AND ABBREVIATED TERMS .......................................................... 9
   3.1. Terms and Definitions ........................................................................................................ 9
   3.2. Abbreviated terms ........................................................................................................... 10

4. OGC API-RECORDS TAILORING ....................................................................................... 13
   4.1. Introduction .................................................................................................................... 13
   4.2. Main use cases and requirements ................................................................................... 14
   4.3. Proposed Interface Design ............................................................................................ 19

5. ASYNCHRONOUS COMMUNICATION FOR OGC API-RECORDS .................................... 29
   5.1. Introduction .................................................................................................................... 29
   5.2. Main Use Cases and Requirements ................................................................................ 29
   5.3. Proposed Interface Design ............................................................................................ 31

6. APPLICATION OF DATA CENTRIC SECURITY ................................................................. 40
   6.1. Introduction .................................................................................................................... 40
   6.2. Main Use Cases and Requirements ................................................................................ 40
   6.3. Proposed Interface Design ............................................................................................ 45

7. SOFTWARE DESIGN ............................................................................................................ 55
   7.1. Software Static Architecture ......................................................................................... 55
   7.2. Interfaces Context ......................................................................................................... 55
   7.3. Software Components Design ....................................................................................... 60

8. FUTURE WORK ..................................................................................................................... 69
Figure A.86 .............................................................................................................. 146
Figure A.87 .............................................................................................................. 146
Figure A.88 .............................................................................................................. 147
Figure A.89 .............................................................................................................. 147
Figure A.90 .............................................................................................................. 147
Figure A.91 .............................................................................................................. 147
Figure A.92 .............................................................................................................. 147
Figure A.93 .............................................................................................................. 147
Figure A.94 .............................................................................................................. 147
Figure A.95 .............................................................................................................. 148
Figure A.96 .............................................................................................................. 148
Figure A.97 .............................................................................................................. 148
Figure A.98 .............................................................................................................. 148
Figure A.99 .............................................................................................................. 149
Figure A.100 .......................................................................................................... 149
Figure A.101 .......................................................................................................... 150
Figure A.102 .......................................................................................................... 151
Figure A.103 .......................................................................................................... 151
Figure A.104 .......................................................................................................... 151
Figure A.105 .......................................................................................................... 152
Figure A.106 .......................................................................................................... 152
Figure A.107 .......................................................................................................... 152
Figure A.108 .......................................................................................................... 152
Figure A.109 .......................................................................................................... 152
Figure A.110 .......................................................................................................... 152
Figure A.111 .......................................................................................................... 152
Figure A.112 .......................................................................................................... 152
Figure A.113 .......................................................................................................... 153
Figure A.114 .......................................................................................................... 154
Figure A.115 .......................................................................................................... 154
Figure A.116 .......................................................................................................... 155
Figure A.117 .......................................................................................................... 155
Figure A.118 .......................................................................................................... 155
Figure A.119 .......................................................................................................... 155
Figure A.120 .......................................................................................................... 155
Figure A.121 .......................................................................................................... 155
Figure A.122 .......................................................................................................... 155
Figure A.123 .......................................................................................................... 159
Figure A.124 .......................................................................................................... 159
Figure A.125 .......................................................................................................... 160
Figure A.126 .......................................................................................................... 160
RECOMMENDATION 19 ..................................................................................................................50
RECOMMENDATION 20 ..................................................................................................................50
RECOMMENDATION 21 ..................................................................................................................72
RECOMMENDATION 22 ..................................................................................................................73
RECOMMENDATION 23 ..................................................................................................................74
EXECUTIVE SUMMARY

This OGC Testbed-18 Engineering Report (ER) describes the results of the Secure, Asynchronous Catalogs Task in the Testbed-18 Catalogs, Filtering, and Moving Features (CMF) thread. This task explored the following.

- How search processes that are supported in a classical OGC Catalogue Service for the Web (CSW) or ISO 19115 environment can be supported through tailoring of the OGC API-Records specification.
- How an asynchronous catalog scenario can be supported in which metadata publishers push new data to catalog instances that lead to new or updated catalog entries and how subscribers are informed about these updates.
- How Data Centric Security (DCS) can be applied in combination with OGC API-Records to allow encrypted delivery and access of catalog metadata between communication partners.

The Engineering Report assumes that the reader has some familiarity with OGC CSW, ISO19115 and the draft OGC API-Records Standards. It summarizes the options investigated and describes in detail the following main achievements.

- The report proposes a tailoring of the OGC API-Records API allowing it to provide similar capabilities as OGC CSW instances in the context of INSPIRE, allowing the option to either continue using the original ISO19115 metadata records or migrating to emerging metadata formats including GeoDCAT-AP with JSON-LD or RDF/XML serialization.
- A comprehensive asynchronous communication model was proposed and subsequently implemented based on a subscription model through a RESTful interface. The proposed design allows for various notification mechanisms including email, WebPush, and callback URL. A detailed state-transition diagram was proposed modeling the different states a subscription can be in, and the HTTP operations, payloads, and status codes required to change states.
- The Data Centric Security (DCS) options available to be combined with the OGC API-Records API were explored in much detail and applied to all interactions, including creation, publication, and updates of metadata records, (synchronous) discovery of metadata records, and also the proposed subscription mechanism for supporting asynchronous communication. The DCS mechanisms proposed support the confidentiality and integrity use cases, can include or not a Key Management System (KMS) to exchange key information among parties, and are fully based on well-known existing IETF standards such as JOSE, JWT, JWE, JWKm and JWS. Examples of all interactions are described in the Engineering Report in full detail. As the proposed solution is fully standards-based, implementations can be realized with a minimum amount of lines of code as open-source libraries supporting the required protocols and formats are available for many languages. The Testbed participant implementations used Python and Javascript for interacting with the various endpoints.
A major result of the activity is the genericity of the proposed solutions. Both the Data Centric Security and the asynchronous communication via subscription models apply equally well to other API, including other OGC (JSON) API or other catalog API including OpenSearch or SpatioTemporal Asset Catalogs (STAC) or other JSON(-LD)-based metadata encodings including GeoDCAT-AP, “OGC EO Dataset Metadata GeoJSON(-LD) Encoding Standard” OGC 17-003r2, “EO Collection GeoJSON(-LD) Encoding” OGC 17-084r1, Service metadata as per OGC 19-020r1, STAC Collection Specification and STAC Item Specification.

During the activity, several areas requiring further work were identified, which are documented in a Future Work chapter of the Engineering Report. The proposed future work items include the following.

- Define a generic OGC API-Notification service and corresponding OGC API Common notification patterns.
- Support detailed notification of (catalog content) changes (incl. deletions) for use by harvesters.
- Apply access tokens with the DPoP (Demonstrating Proof of Possession) header field.
- Combine DCS principles with streaming, e.g., provide JWS/JWE containing a FeatureCollection in chunks.

**KEYWORDS**

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

OGC, API-Records, Catalog, CSW, DCS, ISO-19115, security, web service
SECURITY CONSIDERATIONS

No security considerations have been made for this document.
SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- agentschap Digitaal Vlaanderen, Belgium
- George Mason University, United States of America
- Helyx Ltd., United Kingdom
- Secure Dimensions GmbH, Germany
- Spacebel s.a., Belgium

SUBMITTERS

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>AFFILIATION</th>
<th>ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yves Coene</td>
<td>Spacebel s.a.</td>
<td>Editor</td>
</tr>
<tr>
<td>Christophe Noël</td>
<td>Spacebel s.a.</td>
<td>Editor</td>
</tr>
<tr>
<td>Andreas Matheus</td>
<td>Secure Dimensions GmbH</td>
<td>Contributor</td>
</tr>
<tr>
<td>Eugene Yu</td>
<td>George Mason University</td>
<td>Contributor</td>
</tr>
<tr>
<td>Li Lin</td>
<td>George Mason University</td>
<td>Contributor</td>
</tr>
<tr>
<td>Philip Hawkins</td>
<td>Helyx</td>
<td>Contributor</td>
</tr>
<tr>
<td>Bart Cosyn</td>
<td>agentschap Digitaal Vlaanderen</td>
<td>Contributor</td>
</tr>
<tr>
<td>Loes Deventer</td>
<td>agentschap Digitaal Vlaanderen</td>
<td>Contributor</td>
</tr>
<tr>
<td>Geraldine Nolf</td>
<td>agentschap Digitaal Vlaanderen</td>
<td>Contributor</td>
</tr>
</tbody>
</table>
ABSTRACT

This OGC Testbed-18 Engineering Report (ER) describes the results of the Secure, Asynchronous Catalogs Task in the Testbed-18 Catalogs, Filtering, and Moving Features (CMF) thread. This task explored the following.

- How search processes that are supported in a classical OGC Catalogue Service for the Web (CSW)/ISO 19115 environment can be supported through tailoring of the OGC API-Records specification.

- How an asynchronous catalog scenario can be supported in which metadata publishers push new data to catalog instances that lead to new or updated catalog entries and how subscribers are informed about these updates.

- How Data Centric Security (DCS) can be applied in combination with OGC API-Records to allow encrypted delivery and access of catalog metadata between communication partners.
1

SCOPE
This is the OGC Testbed-18 Engineering Report (ER) documenting the results of the Secure, Asynchronous Catalogs Task. It describes the following.

- How search processes in a classical OGC Catalogue Service for the Web (CSW) or ISO 19115 environment can be supported through tailoring of the OGC API-Records specification.

- Support for an asynchronous catalog scenario in which metadata publishers push new data to catalog instances that lead to new or updated catalog entries and how subscribers are informed about these updates.

- The application of Data Centric Security (DCS) principles in combination with OGC API-Records to allow encrypted and/or signed delivery and access to catalog metadata to support confidentiality and integrity of catalog metadata and search responses.

Introduction

Secure, asynchronous catalogs are contributions to open science environments. Such catalogs help address challenges created by more and more data becoming available and the corresponding complexity of discovery and binding.

This Testbed-18 Engineering Report covers contributions in the following areas.

- With OGC CSW and the emergence of the draft OGC API-Records, users have two standards-based alternatives to interact with catalog services. Whereas OGC CSW implementation instances for ISO 19115 metadata have been used extensively in the past, the OGC API-Records standard is still in development. The draft OGC API-Records standard defines three main interaction building blocks: Record, Collections, and the Records API. The ‘record building block’ defines the core schema of a catalog record. This includes a small number of properties that are common across all resource types and need to be further profiled in order to support ISO 19115:2014 and ISO 19115:2003. Testbed-18 explored how search processes supported in a classical OGC CSW/ISO 19115 environment can be supported using an implementation of OGC API-Records.

- Current OGC APIs define synchronous RESTful web services that return the response on the same socket pair as the one used to receive the request. This interaction pattern works well if the requested response format is streaming-ready and can be delivered immediately. This approach does fail in typical asynchronous interaction scenarios where clients want to subscribe to updates for previously provided queries. Testbed-18 proposed and implemented asynchronous catalog scenarios. In these scenarios, publishers push new metadata to catalog instances that lead to new or updated catalog entries with subscribers being informed about these updates.

- Data Centric Security (DCS) is an approach to apply security directly to the data, independent of security features provided by the network, servers, or applications. For Data Centric Security in the geospatial domain, proof of concept implementations were developed through work in OGC Testbed-15, Testbed-16 and Testbed-17. Testbed-18
extended the developed solutions to work with OGC API-Records to allow encrypted delivery and access of catalog metadata between communication partners.
NORMATIVE REFERENCES
NOMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their
content constitutes requirements of this document. For dated references, only the edition cited
applies. For undated references, the latest edition of the referenced document (including any
amendments) applies.

Open API Initiative: OpenAPI Specification 3.0.2, 2018 https://github.com/OAI/OpenAPI-
Specification/blob/master/versions/3.0.2.md

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: IETF RFC 2616,
Hypertext Transfer Protocol — HTTP/1.1. RFC Publisher (1999). https://www.rfc-
editor.org/info/rfc2616.


info/rfc8288.

editor.org/info/rfc7240.

M. Jones, J. Bradley, N. Sakimura: IETF RFC 7515, JSON Web Signature (JWS). RFC Publisher


editor.org/info/rfc7517.


M. Jones: IETF RFC 7797, JSON Web Signature (JWS) Unencoded Payload Option. RFC Publisher

M. Jones, J. Bradley, H. Tschofenig: IETF RFC 7800, Proof-of-Possession Key Semantics for JSON
rfc7800.


Kellog G., Champin P., Longley D.: W3C JSON-LD 1.1, A JSON-based Serialization for Linked
Data, W3C Recommendation 16 July 2020 https://www.w3.org/TR/json-ld11/


Uwe Voges: OGC 08-197r4, **INSPIRE Conformance Class of OGC Cataloguing of ISO Metadata (CIM) using the ebRIM profile of CS-W**, 2011-10-11


Panagiotis (Peter) A. Vretanos, Clemens Portele: OGC 21-065, Common Query Language (CQL2), https://docs.ogc.org/DRAFTS/21-065.html


3

TERMS, DEFINITIONS, AND ABBREVIATED TERMS
3TERMS, DEFINITIONS, AND ABBREVIATED TERMS

3.1. Terms and Definitions

3.1.1. Access Token
A token that can be provided as part of a service request that grants access to the service being invoked on. This is part of the OpenID Connect and OAuth 2.0 specification [OGC 21-020r1].

3.1.2. Data-Centric Security
Data-centric security emphasizes the dependability of the data itself rather than the security of networks, servers, or applications [Wikipedia].

3.1.3. (Spatial) Data Set
Identifiable collection of (spatial) data [INSPIRE].

3.1.4. (Spatial) Data Set Series
Collection of (spatial) data sets sharing the same product specification [INSPIRE].

3.1.5. Discovery Service
Service that makes it possible to search for spatial data sets and services on the basis of the contents of the corresponding metadata and to display the contents of the metadata [INSPIRE].

3.1.6. Granule
A granule is the finest granularity of data that can be independently managed. A granule usually matches the individual file of EO satellite data [CEOS-BP].
3.1.7. STANAG

In NATO, Standardization Agreement defines processes, procedures, terms, and conditions for common military or technical procedures or equipment between the member countries of the alliance.

3.2. Abbreviated terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Application Package</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CEOS</td>
<td>Committee on Earth Observation Satellites</td>
</tr>
<tr>
<td>CQL</td>
<td>Common Query Language</td>
</tr>
<tr>
<td>CSW</td>
<td>Catalogue Service for the Web</td>
</tr>
<tr>
<td>DCAT</td>
<td>Data Catalog Vocabulary</td>
</tr>
<tr>
<td>DCS</td>
<td>Data Centric Security</td>
</tr>
<tr>
<td>DEK</td>
<td>Data Encryption Key</td>
</tr>
<tr>
<td>DPoP</td>
<td>Demonstrating Proof-of-Possession</td>
</tr>
<tr>
<td>EO</td>
<td>Earth Observation</td>
</tr>
<tr>
<td>EP</td>
<td>Extension Package</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IdP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>INSPIRE</td>
<td>INfrastructure for SPatial InfoRmation in Europe</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>JOSE</td>
<td>Javascript Object Signing and Encryption</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JWE</td>
<td>JSON Web Encryption</td>
</tr>
<tr>
<td>JWK</td>
<td>JSON Web Key</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>JWS</td>
<td>JSON Web Signature</td>
</tr>
<tr>
<td>JWS/CT</td>
<td>JSON Web Signature Clear Text</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>KEK</td>
<td>Key Encryption Key</td>
</tr>
<tr>
<td>KMS</td>
<td>Key Management System</td>
</tr>
<tr>
<td>OAI-PMH</td>
<td>Open Archive Initiative Protocol for Metadata Harvesting</td>
</tr>
<tr>
<td>OGC</td>
<td>Open Geospatial Consortium</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>STAC</td>
<td>SpatioTemporal Asset Catalog</td>
</tr>
<tr>
<td>STANAG</td>
<td>Standardization Agreement</td>
</tr>
<tr>
<td>SWG</td>
<td>Standard Working Group</td>
</tr>
<tr>
<td>TIE</td>
<td>Technology Integration Experiment</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>

**OPEN GEOSPATIAL CONSORTIUM 22-018**
4

OGC API-RECORDS TAILORING
4.1. Introduction

This chapter includes the following.

1. Whether OGC API – Records can support classical discovery workflows comparable with OGC CSW/ISO 19115 setups.
2. A proposed tailoring based on the use cases and requirements gathered during the project.

4.1.1. CSW ISO AP / ISO19115 Setups

OGC CSW has been used for ISO19115 metadata (and in particular its XML encoding ISO19139(-2)) extensively in the past. The CSW ISO Application Profile (AP) OGC 07-045r1 is widely implemented in Europe as the INSPIRE discovery service as one of the INSPIRE Network Services [3]. In addition, the INSPIRE community uses ISO19115 metadata and its XML encodings (e.g., ISO19139(-2)) for encoding dataset, dataset series, and services as explained in the Technical Guidance document [4]. The Joint Research Centre (JRC) of the European Commission (EC) are proposing a GeoDCAT-AP encoding as a Resource Description Framework (RDF) vocabulary with information content covering the union of metadata elements of the core profile of ISO 19115:2003 and those defined in the framework of the INSPIRE Directive of the European Union. This allows for a JSON-LD encoding for metadata records currently described within the INSPIRE community.

4.1.2. CSW ebRIM AP / ISO19115 Setups

In addition to CSW ISO AP, other communities use the CSW ebRIM Application Profile (AP) OGC 07-114r4 for discovery of resources described using the same ISO19115-based metadata. A dedicated ebRIM Extension Package for CSW ebRIM was developed by the OGC members and released in 2014 (OGC 13-084r2). This standard is known as I15 Extension Package (EP) or Cataloguing of ISO19115 Metadata (CIM) EP. The extension package was originally used by the Earth Observation (EO) community to support EO collection, service and sensor discovery as described in OGC 11-035r1. A typical EO discovery workflow consists of two steps: An initial search for EO collections (similar to INSPIRE dataset series) followed by a search for EO granules (i.e., products, similar to INSPIRE datasets) belonging to the collection. The search parameters available for the granule search typically differ according to the collection, such as filtering according to cloud cover percentage only applies to collections from optical sensors. While discovery services based on the ebRIM CSW service bindings have been replaced by OpenSearch service bindings in the EO community (See CEOS-BP and OGC 13-026r9), many
agencies still describe their EO collections using ISO19139(-2) metadata records. The /gmd:MD_Metadata/gmd:parentIdentifier property (ISO19139:2007), /eop:metaDataProperty/eop:EarthObservationMetaData/eop:parentIdentifier (OGC 10-157r4) property, $.properties.parentIdentifier (OGC 17-003r2) property and eo:parentIdentifier queryable (OGC 13-026r9) all reflect the hierarchical relation between the metadata record of an EO granule and the metadata record of the corresponding EO collection.

The gmd:parentIdentifier relation is also mandatory in the INSPIRE context if a higher level dataset is available in the hierarchy [5][6].

### 4.2. Main use cases and requirements

**Table 1 — User stories — discovery workflow**

<table>
<thead>
<tr>
<th>User Story</th>
<th>As a ...</th>
<th>I want to ...</th>
<th>so that I can ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Metadata Consumer</td>
<td>obtain metadata about the catalog service endpoint, including its service bindings.</td>
<td>interact with the catalog.</td>
</tr>
<tr>
<td>2</td>
<td>Metadata Consumer</td>
<td>access the catalog service and perform requests for catalog records.</td>
<td>discover metadata records available in the catalog.</td>
</tr>
<tr>
<td>2.1</td>
<td>Metadata Consumer</td>
<td>access the catalog service and perform search requests for catalog records matching my search criteria.</td>
<td>discover metadata records available in the catalog.</td>
</tr>
<tr>
<td>2.2</td>
<td>Metadata Consumer</td>
<td>access the catalog service and retrieve one specific complete catalog record in a specific format.</td>
<td>retrieve all detailed metadata information for a single record in my preferred format (XML, ISO, RDF, ...).</td>
</tr>
<tr>
<td>3</td>
<td>Metadata Publisher</td>
<td>upload metadata records in the catalog</td>
<td>make them discoverable for metadata consumers.</td>
</tr>
<tr>
<td>3.1</td>
<td>Metadata Publisher</td>
<td>update or remove metadata records available in the catalog</td>
<td>perform metadata changes.</td>
</tr>
<tr>
<td>3.2</td>
<td>Metadata Publisher</td>
<td>obtain metadata records available in the catalog.</td>
<td>perform metadata changes.</td>
</tr>
</tbody>
</table>

Clause 4.1 of [3] lists the identified use cases that an INSPIRE Discovery Service has to fulfill to support the typical INSPIRE workflow. Such a service is known as the INSPIRE Profile of CSW ISO AP. The corresponding use case diagram is depicted below.
The following subsections assess whether OGC API-Records can satisfy each of these use cases and derive requirements for the additional tailoring of OGC API-Records.

### 4.2.1. Use Case 1 (UC1): Get Discovery Service Metadata

A CSW catalog allows clients to retrieve service metadata from a server via the `GetCapabilities` operation which may contain a `MetadataURL` element (Req. 7 and Table 3 of [3]).

Similarly, an OGC API-Records implementation should provide the same information in the following way, to support the same workflow:

#### Table 2 — Discovery service metadata comparison

<table>
<thead>
<tr>
<th>Extended CSW Metadata</th>
<th>OGC API-Records Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service identification, Service provider, Operations metadata</td>
<td>Landing Page link rel=&quot;describedby&quot; and type=&quot;application/vnd.iso.19139+xml&quot; and /conformance endpoint. The <code>hreflang</code> attribute can be used in the link object to provide the discovery service metadata in multiple languages (Req. 6 of [3]).</td>
</tr>
<tr>
<td>Filter capabilities</td>
<td>Landing Page link rel=&quot;queryables&quot;, type=&quot;application/schema+json&quot; and /queryables endpoint as per OGC 19-079r1. Similarly, CQL capabilities are to be advertised.</td>
</tr>
</tbody>
</table>
RECOMMENDATION 1

STATEMENT: The discovery interface should provide discovery service metadata equivalent to the above Extended CSW Capabilities information.

4.2.2. Use Case 2 (UC2): Discover Metadata

To implement the INSPIRE workflow, the CSW ISO AP supports the GetRecords operation as per clause 4.3.2 of [3] with following two additional parameters.

- The language parameter: The OGC API-Records API server can provide a similar capability by supporting the usual HTTP content negotiation mechanisms and process the Accept-Language header of a request (as per clause 7.10 of OGC 17-069r3).

- The query parameter: Supported by the filter statement of CSW ISO AP. To support similar Filter_Capabilities as can be advertised in a CSW Capabilities document, the OGC API-Records server will need to implement a subset of the draft OGC Common Query Language (CQL2) OGC 21-065 and support CQL2 filter expressions at the /collections/{collection-id}/items endpoint.

NOTE: INSPIRE requires the Discovery Service to advertise the default language in the CSW GetCapabilities response. Proposing a similar mechanism to advertise the default language is further work. Possible approaches include:

- advertise default language in the OGC API-Records Landing Page response;
- advertise default language in the OGC API-Records API Definition response; and
- use of hreflang in the link object referring to the search endpoint (/collections/{id}/items) in a Collection response.

RECOMMENDATION 2

STATEMENT: The discovery interface should support the equivalent of the (INSPIRE) search parameters defined in clause 4.4 of [3].

When past OGC discovery interfaces were defined, INSPIRE requirements were taken into account by proposing the search parameters required by the INSPIRE discovery service. The OGC OpenSearch standard OGC 13-026r9 was tailored in this way to cover INSPIRE requirements and a dedicated INSPIRE conformance class was added. The CSW CIM EP specification was also tailored for INSPIRE requirements.

To ensure a common response format, a GetRecords request is recommended to support “application/xml” and ISO19139 as outputSchema (outputSchema=http://www.isotc211.org/2005/gmd).
RECOMMENDATION 3
STATEMENT The discovery interface should support access to the (original) XML-based metadata (i.e., ISO19139, ISO19139-2, ISO19115-3, etc.) through content negotiation or indirectly via an $.properties.links[] object with rel="via" in a regular API-Records GeoJSON search response.

RECOMMENDATION 4
STATEMENT The discovery interface should use the media types listed in OGC13-026r9 for identifying ISO19139, ISO19139-2, or ISO19115-3 representations, i.e., “application/vnd.iso.19139+xml”, “application/vnd.iso.19139-2+xml”, and “application/vnd.iso.19115-3+xml” for content negotiation or as type attribute of the rel="via" links.

An INSPIRE discovery service has to advertise all its queryables. To achieve this capability, the OGC API-Records catalog needs to advertise its queryables via the appropriate endpoints /queryables and/or /collections/{collection-id}/queryables.

RECOMMENDATION 5
STATEMENT The discovery interface should support discovery of "datasets," "dataset series," and "services".

RECOMMENDATION 6
STATEMENT The discovery interface should support GeoDCAT-AP or DCAT-AP encoding as primary metadata representation.

RECOMMENDATION 7
STATEMENT The discovery interface should support retrieval of records using their GeoDCAT-AP or DCAT-AP encoding in at least the following RDF serialisations: application/rdf+xml and application/ld+json. Support for text/turtle is optional.
4.2.3. Use Case 3 (UC3): Publish Metadata

The Publish Metadata operation supports editing (insert, update, and delete) of metadata elements of resources in the catalog (push or pull metadata mechanisms). To support the INSPIRE workflow, the CSW ISO AP has to support the Transaction (push) or Harvest (pull) operation as per clause 4.3.3 of [3].

To support similar capabilities, the OGC API-Records server will need to implement the draft “OGC API — Features — Part 4: Create, Replace, Update, and Delete” OGC 20-002 standard.

NOTE The above OGC 20-002 standard may also support editing collections available at a /collections endpoint when the resources endpoint is interpreted as [landingPageUri]/collections. This case is similar to the [landingPageUri]/processes case (Example 2).

**RECOMMENDATION 8**

**STATEMENT** The catalog interface should support insertion, deletion, and update of "datasets," "dataset series," and "services" metadata records.

**RECOMMENDATION 9**

**STATEMENT** The catalog interface should support insertion and updates of records using their original ISO19139 or ISO19115-3 encoding.

**RECOMMENDATION 10**

**STATEMENT** The catalog interface should support insertion and updates of records using their GeoDCAT-AP or DCAT-AP encoding.

**RECOMMENDATION 11**

**STATEMENT** The catalog interface should support insertion and updates of records using their GeoDCAT-AP or DCAT-AP encoding in at least the following RDF serialisations: application/rdf+xml and application/ld+json. Support for text/turtle is optional.
4.3. Proposed Interface Design

In clause 6.5, the draft OGC API-Records standard defines three building blocks that each can be tailored for a specific community. The following subsections propose a tailoring of each of these building blocks to support the requirements that were identified in the previous section.

### REQUIREMENT 1

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/iso/record-types-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>The catalog shall support discovery resources of type series (if applicable) via one of the following mechanisms.</td>
</tr>
<tr>
<td>A</td>
<td>(Option 1) Via the /collections and /collections/{resource-id} endpoints.</td>
</tr>
<tr>
<td>B</td>
<td>(Option 2) Via the /collections/{collection-id}/items and /collections/{collection-id}/items/{resource-id} endpoints.</td>
</tr>
</tbody>
</table>

### REQUIREMENT 2

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/iso/record-types-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>The catalog shall support discovery resources of type dataset or service (if applicable) via the /collections/{collection-id}/items and /collections/{collection-id}/items/{resource-id} endpoints.</td>
</tr>
</tbody>
</table>

As explained above, there are two alternative ways to map series on the data model. The tailoring of the draft OGC API-Records Standard and interpretation of the requirements below depend on the selected mapping. The term series search endpoint is used to indicate the search endpoint for series search selected according to the above requirement. The terms dataset and services search endpoint apply to the search endpoints for dataset and service records (if applicable).

#### 4.3.1. Record Collection (Catalog) Tailoring

Section 8 of OGC 20-004 defines the schema for the collection resource as an extension of the collection schema defined in OGC API-Features and OGC API-Common — Part 2: Geospatial Data.
RECOMMENDATION 12

If additional GeoJSON properties are required in the catalog collection schema to represent ISO19139(-2) or ISO19115-3 dataset series metadata properties, they should be borrowed from the GeoDCAT-AP or DCAT-AP encoding when possible. Applicable to Option-1 only.

RECOMMENDATION 13

The catalog should support the ISO19139(-2) or ISO19115-3 metadata representations for the collection schema (via content negotiation or a query parameter, e.g., f=). This represents an additional requirements class in addition to “clause 10.3 Requirements Class GeoJSON” and “clause 10.2 Requirements Class HTML” for the catalog collection schema in OGC 20-024. Applicable to Option-1 only.

4.3.2. Record Schema Tailoring

Section 16.1 of OGC 20-004 defines the GeoJSON (application/geo+json) encoding of the catalog record schema.

RECOMMENDATION 14

If additional GeoJSON properties are required in the catalog record schema to represent ISO19139(-2) or ISO19115-3 metadata properties, they should be borrowed from the GeoDCAT-AP or DCAT-AP encoding when possible.

NOTE The EO Collection GeoJSON(-LD) Encoding (OGC 17-084r1) Best Practice proposes a reusable GeoJSON encoding with corresponding mapping to ISO19139(-2) and NASA UMM-C properties in its Annex C, consistent with GeoDCAT-AP via a JSON-LD context definition.

RECOMMENDATION 15

The catalog should support the ISO19139(-2) or ISO19115(-3) metadata representations for the record schema (via content negotiation or a query parameter, e.g., f=). This represents an additional requirements class in addition to “clause 16.1 Requirements Class GeoJSON” and “clause 16.3 Requirements Class HTML” for the catalog record schema.

4.3.3. Records API Tailoring
## REQUIREMENT 3

**LABEL** /req/iso/type-queryable

**STATEMENT** The OGC API-Records search endpoint shall support the queryable type, equivalent to the CSW queryable Type (MD_Metadata.hierarchyLevel) to filter (ISO19139) metadata records based on the hierarchyLevel (i.e., series, dataset, service) (See OGC 07-045r2).

Examples:
- [https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=series](https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=series)
- [https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=dataset](https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=dataset)
- [https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=service](https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items?type=service)

To support the search workflows using search criteria defined in Table 4 of [3], the following requirements apply.

## REQUIREMENT 4

**LABEL** /req/iso/mandatory-series-queryables

**STATEMENT** The OGC API-Records series search endpoint shall support the following additional search parameters with expected data type, meaning and mapping on (ISO19139) metadata as defined in OpenSearch Extension for Earth Observation (OGC 13-026r9).

<table>
<thead>
<tr>
<th>A</th>
<th>useLimitation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>accessConstraint</td>
</tr>
<tr>
<td>C</td>
<td>otherConstraint</td>
</tr>
<tr>
<td>D</td>
<td>classification</td>
</tr>
<tr>
<td>E</td>
<td>organisationName</td>
</tr>
<tr>
<td>F</td>
<td>organisationRole</td>
</tr>
<tr>
<td>G</td>
<td>topicCategory</td>
</tr>
<tr>
<td>H</td>
<td>lineage</td>
</tr>
</tbody>
</table>
REQUIREMENT 4

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>keyword</td>
</tr>
<tr>
<td>J</td>
<td>denominator</td>
</tr>
<tr>
<td>K</td>
<td>distanceValue</td>
</tr>
<tr>
<td>L</td>
<td>distanceUOM</td>
</tr>
<tr>
<td>M</td>
<td>Accept-Language (header) — i.e., metadata language.</td>
</tr>
<tr>
<td>N</td>
<td>title</td>
</tr>
<tr>
<td>O</td>
<td>abstract</td>
</tr>
<tr>
<td>P</td>
<td>specificationTitle</td>
</tr>
<tr>
<td>Q</td>
<td>specificationDate</td>
</tr>
<tr>
<td>R</td>
<td>specificationDataType</td>
</tr>
<tr>
<td>S</td>
<td>degree</td>
</tr>
</tbody>
</table>

REQUIREMENT 5

| LABEL | /req/iso/mandatory-item-queryables |
| STATEMENT | The OGC API-Records /collections/{collectionId}/items endpoint (a.k.a. dataset and service search endpoint) shall support the additional search parameters listed in the previous requirement /req/iso/mandatory-series-queryables. |

REQUIREMENT 6

| LABEL | /req/iso/advertise-series-queryables |
| STATEMENT | The OGC API-Records interface shall advertise the available queryables for the series search endpoint (if applicable) at the endpoint corresponding the option selected for "series" discovery (See above) |
|   | Via the /queryables endpoints (If Option 1 was selected). |
**REQUIREMENT 6**

B Via the /collections/{collection-id}/queryables (if Option 2 was selected).

**REQUIREMENT 7**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/iso/advertise-item-queryables</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>The OGC API-Records interface shall advertise the available queryables for the dataset and service search endpoints (if applicable) at the /collections/{collection-id}/queryables endpoint.</td>
</tr>
</tbody>
</table>

The example below provides an example of a /queryables response advertising search parameters required for a typical INSPIRE discovery workflow.

```json
{
  "title": "Available search parameters",
  "type": "object",
  "properties": {
    "limit": {
      "title": "limit",
      "type": "integer"
    },
    "bbox": {
      "title": "bbox",
      "type": "array",
      "minItems": 4,
      "maxItems": 6,
      "items": {
        "type": "number"
      }
    },
    "datetime": {
      "title": "datetime",
      "type": "string"
    },
    "externalId": {
      "title": "externalId",
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "type": {
      "title": "type",
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "q": {
      "title": "q",
      "type": "array",
      "items": {
        "type": "string"
      }
    },
```

OPEN GEOSPATIAL CONSORTIUM 22-018
"items": {
  "type": "string"
}
,"useLimitation": {
  "title": "useLimitation",
  "type": "string"
},
"accessConstraint": {
  "title": "accessConstraint",
  "type": "string",
  "enum": [
    "copyright",
    "patent",
    "trademark",
    "license",
    "intellectualPropertyRights",
    "restricted",
    "otherRestrictions"
  ]
},
"otherConstraint": {
  "title": "otherConstraint",
  "type": "string"
},
"classification": {
  "title": "classification",
  "type": "string",
  "enum": [
    "unclassified",
    "restricted",
    "confidential",
    "secret",
    "topSecret"
  ]
},
"organisationName": {
  "title": "organisationName",
  "type": "string",
  "enum": [
    "CEDA",
    "CMEMS",
    "DE/DLR",
    "ESA/ESRIN",
    "FR/CNES",
    "JP/JAXA/EOC",
    "VITO"
  ]
},
"organisationRole": {
  "title": "organisationRole",
  "type": "string",
  "enum": [
    "resourceProvider",
    "custodian",
    "owner",
    "user",
    "distributor",
    "originator",
    "pointOfContact",
    "principalInvestigator",
    "processor",
    "publisher",
    "processor"
  ]
}
Figure 3 — Advertising INSPIRE Search Parameters Via the /queryables Response

### REQUIREMENT 8

**LABEL**  
/req/iso/update-series

**STATEMENT**  
The OGC API-Records shall implement (series) metadata record insertion, update, and deletion at the /collections (Option 1) or /collections/{collection-id}/items endpoint (Option 2) using the paths and HTTP methods as per OGC20-002.

### REQUIREMENT 9

**LABEL**  
/req/iso/update-items

**STATEMENT**  
The OGC API-Records shall implement (dataset and service) metadata record insertion, update, and deletion at the /items endpoint using the paths and HTTP methods as per OGC20-002.

See Annex A Example 3.3 for a detailed [Jupyter Notebook](https://tb18.cat.org/collections/series/queryables) example using the POST operation and application/xml media type to insert a metadata record and the DELETE operation to remove the record.

The resource definition diagram below summarizes the available resources, the applicable HTTP methods, available representations, and the paths for each of the resources. Some parts of the diagram only apply when Option 1 was used for modeling (ISO) series, i.e., when /collections/{collection-id} represents a series.
Figure 4 — Resource Definition Diagram for Tailored OGC API-Records
ASYNCHRONOUS COMMUNICATION FOR OGC API-RECORDS
ASYNCHRONOUS COMMUNICATION FOR OGC API-RECORDS

5.1. Introduction

One of the goals for Testbed-18 was to contribute to asynchronous communication discussions by implementing an asynchronous catalog. The OGC Testbed-18 Call for Participation (CFP) describes asynchronous communication as scenarios where clients want to subscribe to updates for previously provided queries. Those subscribers should be informed of new or updated catalog entries (e.g., inserted by a publisher).

The designed solution described below is based on the **Subscription** resource which represents the mission of delivering update notification based on the requested records query and the provided delivery method.

Although the approach above does not specifically tackle the concern of queries that cannot be processed within regular HTTP time out (e.g., when the server needs encrypting), such a scenario is simply a special case of a subscription with a single update notification (when the response is available).

5.2. Main Use Cases and Requirements

Table 3 — User Stories – Asynchronous Catalog

<table>
<thead>
<tr>
<th>User Story</th>
<th>As a …</th>
<th>I want to …</th>
<th>so that I can …</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Metadata Consumer</td>
<td>subscribe to a search request.</td>
<td>receive additional search results in the future matching my original search criteria.</td>
</tr>
<tr>
<td>22</td>
<td>Metadata Consumer</td>
<td>unsubscribe to a search request subscription.</td>
<td>stop receiving additional search results in the future.</td>
</tr>
<tr>
<td>23</td>
<td>Metadata Consumer</td>
<td>define the frequency with which I want to receive additional search results (e.g., daily).</td>
<td>receive search result updates grouped by period.</td>
</tr>
<tr>
<td>24</td>
<td>Metadata Consumer</td>
<td>provide an HTTP endpoint or email address to receive notification of additional search results</td>
<td>choose the notification mechanism.</td>
</tr>
<tr>
<td>25</td>
<td>Metadata Consumer</td>
<td>indicate at subscription time that the server shall apply Data Centric Security (signature, encryption) to the future search responses.</td>
<td>choose the security level for future additional search results.</td>
</tr>
</tbody>
</table>

The subscription interface should allow a client to subscribe to the results (and updates) of a catalog query. The subscription implies that the catalog must notify (asynchronously) the subscribers of subsequent modifications matching the query until the user unsubscribes.

In particular, the following reports and interfaces covering asynchronous interactions were reviewed and considered for the design of a solution.

- The asynchronous support capability provided in the draft OGC API – Processes Standard involves the polling of status (supported by the Core requirements class) and the callback-driven approach (supported by the Callback requirement class) which relies on OpenAPI native capabilities (https://swagger.io/docs/specification/callbacks/).
- The behavior models described in the OGC Publish-Subscribe White Paper (OGC 20-081) include the Delayed Response, the Standing Request, the Synchronization and the Publish-Subscribe.
- The Geosynchronization mechanism specified in the OGC OWS 7 Engineering Report — Geosynchronization service (OGC 10-069r2) describing the Subscriptions, Topics, and Publishers resources and related operations.
- The Publish/Subscribe Interface Standard (OGC 13-131r1) that supports the core components and concepts of the Publish/Subscribe message exchange pattern with OGC Web Services.
- The OGC Testbed-13 Asynchronous Services ER (OGC 17-028) that essentially describes the Publish-Subscribe and the status polling of the legacy OGC Web Processing Service interface.

As a starting point, the OGC API — Processes Callback approach was adapted to the API Records context as follows.

- The behavior model is extended to support a subscriber requesting continuous update notifications about new catalog records.
- The subscriber must provide a notification expiration time to avoid flooding a subscriber with too many callback notifications.
- The Dismiss operation must support the cancellation of the subscription.
- The content of the notification may either include the (updated) records or a link to these records.
While the callback could be specified in HTTP GET Catalog request, the Testbed participants reached a consensus to design a blank solution and handle the subscription creation from a dedicated HTTP POST on the resource endpoint.

### 5.3. Proposed Interface Design

The following subsections propose a OGC Subscription requirement class for the OGC API-Records draft standard. The proposed interface relies on subscription resources. A subscription is a job that delivers notifications when a client subscribes to resources updates (i.e., new records or modified records).

The following flow applies to a subscription and the resulting notifications.

- The Catalog Client (on the behalf of a signed user) submits a subscription request to the Catalog server and provides the URL to the subscribed resources and the delivery options (delivery method, expiration, schedule).
- The Catalog server registers the subscription and returns HTTP code 201 along with the link to the created subscription.
- Each time the server schedules a check on the subscribed records, a notification is prepared and sent to the configured receiver.

![Asynchronous Pattern for API Records](image)

**Figure 5 — Asynchronous Pattern for API Records**

The interface provides operations to create, read, update, and delete subscriptions. The resource definition diagram below summarizes the applicable HTTP methods and the paths for each of the resources.
5.3.1. Subscription Resource Schema

A subscription is represented as the following set of properties.

- **Resources-uri**: Identifier of the resources to which the user subscribes (typically a URL to catalog records).
- **Delivery**: Method and target for notification deliveries formatted as protocol:address[,additional parameters] (e.g., mailto:you@ogc.org).
- **Expires**: Seconds the subscription is valid (or ISO 8601 date to be discussed)/
- **Schedule**: Defines the notification schedule using a cron-like format (e.g., “0 0 * * 0”).
- **Status**: Indicates the state of a created subscription (e.g., started, completed, cancelled).
- **Deliveries**: List the URL links to the resources that have already been delivered (in a notification).

Additional DCS properties (public-key, wrapped-dek) are detailed further.

The OpenAPI 3.0 schema `subscription.yaml` is represented on the class diagram below.
5.3.2. Create a Subscription

A client can create a subscription by identifying the resources for which update notifications should be sent and indicating the delivery options.

**REQUIREMENT 10**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/subscriptions/create</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>The server shall support the HTTP POST operation at the path /subscriptions. The request shall be based on the SubscriptionRequest class of the subscription.yaml schema.</td>
</tr>
</tbody>
</table>

**REQUIREMENT 11**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/subscriptions/create-success</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>A successful execution of the operation shall be reported as a response with a HTTP status code 201. The header of the response shall return the HTTP Location header that contains a link to the newly created subscription.</td>
</tr>
</tbody>
</table>

The example below illustrates a typical subscription request.

```
Figure 8 — Subscription Request Example

See also Annex A Example 5.1.

Alternatively, a subscription could be requested directly when retrieving a list of catalog records. This could be achieved by extending the HTTP GET request with the relevant parameters (such as the OGC API Processes callbacks). However, the Testbed participants agreed to focus on the specific subscription endpoint in the context of the OGC API Records.

### 5.3.3. Receive subscription notifications

If the subscriber provides a delivery method, then the results are communicated to the delivery endpoint in accordance with the delivery options (expiry, schedule, etc.). Also, the subscriber can specify preferences about the notification content (updated records or a link to those records) using the HTTP Prefer header as detailed below.

#### REQUIREMENT 12

**LABEL**

/req/subscription/notification

**STATEMENT**

The server shall deliver notifications when subscribed resources (identified in `resources-uri`) are updated (new resources or modified resources). If the request is accompanied with the HTTP Prefer header then the server SHOULD honor that preference.

**A**

The notification must contain a link to the resources if the subscription request is provided with the Prefer header with value 'return=minimal' or if the header is not present.

**B**

The notification must contain the list of resources if the request is accompanied with the HTTP Prefer header asserting a return=representation preference.

#### REQUIREMENT 13

**LABEL**

/req/subscriptions/notification-delivery

**STATEMENT**

The server shall send update notifications only if a delivery method is provided in the subscription. If provided, the server shall deliver notifications using the method and protocol provided in the delivery property (e.g., mailto:you@ogc.org)
5.3.4. Retrieve a Subscription

A subscription can be retrieved to monitor the subscription status (e.g., started, completed, cancelled) and get links to the updated resources clusters that were reported by the server.

When no delivery method is specified, the subscription plays the same role as the job status document defined in OGC API Processes: The subscription can be polled for discovering updated resources.

REQUIREMENT 16

LABEL /req/subscriptions/subscription-get

STATEMENT The server shall support the HTTP GET operation at the path /subscriptions/{sub_id} to retrieve an existing subscription.

REQUIREMENT 17

LABEL /req/subscriptions/subscription-get-response

STATEMENT A successful execution of the operation shall be reported as a response with a HTTP status code 200. The content of that response shall be based upon the Subscription class of the OpenAPI 3.0 schema subscription.yaml. The deliveries array provide links to delivery units (i.e., a set of updated resources that have been prepared by the server).
RECOMMENDATION 16

STATEDMENT

The catalog should prepare the delivery units following the frequency specified in the schedule property. If an update is not detected, then no delivery unit should be prepared, and no link should be added to the deliveries array.

The example below illustrates the response of the HTTP GET request.

Figure 9 — Get Subscription Response Example

5.3.5. Update and Delete Subscription

A subscription might be deleted using HTTP Delete or can be updated using HTTP Patch for editing any of the subscription property (e.g., delivery, schedule, public-key).

Note that the possibility to update the status of the subscription (e.g., paused) has been raised but not implemented in the scope of the project.

See also Annex A Example 5.6.

REQUIREMENT 18

LABEL /req/subscriptions/subscription-patch

STATEDMENT

The server shall support the HTTP PATCH operation at the path /subscriptions/{sub_id}. The request shall be based on the SubscriptionRequest class of the subscription.yaml schema, and the delivery process should be updated accordingly. The response of a successful PATCH request is 204: Done.

REQUIREMENT 19

LABEL /req/subscriptions/subscription-delete

STATEDMENT

The server shall support the HTTP DELETE operation at the path /subscriptions/{sub_id}. The corresponding subscription should be removed and can no longer be retrieved. The response of a successful DELETE request is 204: Done.
5.3.6. Subscription States

A tailoring of the OGC Subscription requirement was implemented for requesting an explicit subscription status update. The approach extends the existing status (started, completed, cancelled) with additional values, and proposes providing the PATCH operation to update the status of a subscription.

The subscription states as shown on diagram below include the following.

- **created**: The subscription has been submitted by the client.
- **approved**: The subscription endpoint has been approved (with the security code).
- **ready**: The target resources (uri) are provided.
- **started**: The subscription is manually started.
- **stopped**: The subscription is manually stopped.
- **deleted**: The subscription is deleted.

See also Annex A Example 5.4 and Example 5.5.

---

**Figure 10 — Subscription State Diagram**

- POST /subscriptions
  - "name"
  - "resources-uri"
  - "expires"
  - "schedule"
  - "delivery"
  - "security"
  - 201: Location /subscriptions/[id]

- PATCH /subscriptions/[id]
  - "state": "approval"

- PATCH /subscriptions/[id] (["resources-uri": "uri"]
  - "code": $code)

- GET /subscriptions/[id]
  - 410: Gone

- Notes:
  - Id cannot be changed
  - Response for all successful PATCH requests is "204: Done"
  - "": auto transition if "resources-uri" is set

- PATCH /subscriptions/[id]
  - "state": "start"
  - 200: OK

- PATCH /subscriptions/[id]
  - "state": "stop"

- DELETE /subscriptions/[id]
  - 204: Done

- DELETE /subscriptions/[id]
The diagram reflects only explicit operations, but automated status changes might be relevant (e.g., automatically starting the subscription). Moreover, alternative flows and states can be considered such as completed for an expired subscription.
APPLICATION OF DATA CENTRIC SECURITY
6. Application of Data Centric Security

6.1. Introduction

Applying Data Centric Security can be applied to the following interactions.

- Synchronous Search Responses
- Asynchronous Responses
- Metadata Insertions/Updates in the Catalog

6.2. Main Use Cases and Requirements

The table below summarizes the main use cases.

**Table 4 — User Stories — Data Centric Security**

<table>
<thead>
<tr>
<th>User Story</th>
<th>As a ...</th>
<th>I want to ...</th>
<th>So that I can ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Metadata Publisher</td>
<td>upload plain text metadata records in the catalog.</td>
<td>make them discoverable for metadata consumers.</td>
</tr>
<tr>
<td>12</td>
<td>Metadata Publisher</td>
<td>upload signed metadata records in the catalog.</td>
<td>ensure integrity.</td>
</tr>
<tr>
<td>13</td>
<td>Metadata Publisher</td>
<td>upload encrypted metadata records in the catalog.</td>
<td>ensure confidentiality and make it available to only selected metadata consumers.</td>
</tr>
<tr>
<td>14</td>
<td>Metadata Publisher</td>
<td>upload metadata records in the catalog combined with corresponding access rules.</td>
<td>ensure appropriate access rights.</td>
</tr>
<tr>
<td>15</td>
<td>Metadata Consumer</td>
<td>choose whether to receive metadata records in search responses as plain records, signed or encrypted assuming I have the appropriate access rights.</td>
<td>select the appropriate security level.</td>
</tr>
<tr>
<td>16</td>
<td>Metadata Consumer</td>
<td>verify the signature of a metadata record part of a search response.</td>
<td>check integrity and authenticity.</td>
</tr>
<tr>
<td>17</td>
<td>Metadata Consumer</td>
<td>decrypt an encrypted metadata record which is part of a search response.</td>
<td>process, display, etc. the metadata record content.</td>
</tr>
</tbody>
</table>
6.2.1. Synchronous Responses With DCS

The API endpoints /collections/{id}/items (for item search), /collections/{id}/items/{id} (for collection search) etc.) that are subject to encrypted payloads or responses can vary. Clients can discover this through the media types supported for a specific resource (i.e. application/dcs+geo or application/jose).

In the context of an OGC API – Records implementation, the application of DCS has many similarities to the second scenario described in “Testbed-17: Data Centric Security ER” (21-020r1), which depicts the exchange of DCS-protected content from an OGC API – Features implementation. The same ER describes which metadata fields in a response allow a client to successfully decrypt the information.

6.2.1.1. Application/DCS+GEO

According to clause 4.1 of 21-020r1, for OGC API – Features, the DCS data format can be applied as an extension to XML and JSON. Although the ISO19115 tailoring of OGC API – Records may decide to also use XML formats (see above), the focus was on a JSON approach. The response formats proposed for OGC API – Features correspond to the encrypted features in a JSON container (clause 5.2.2 of 21-020r1), based on the schema illustrated below.

![Figure 11 — API Features JOSE Based Containers JSON Schema (dcs+geo)](image)

The potentially sensitive information in the data_description element can be requested with three levels of security (plain, integrity, or confidentiality) depending on the requested media type profile for application/dcs+geo. The catalog server detects that a client wants an encrypted response when it receives a request (via content negotiation) for the application/dcs+geo media type.
When provided in a JSON Web Encryption (JWE) token, the data_description element (see schema above) is encrypted with the public key of the user (i.e., client) providing the public Key Encryption Key (KEK) id previously registered at the Key Management Service (KMS).

The data is always encrypted in a JWE token using a Data Encryption Key (DEK). The kid and kurl of the DEK are included the JWE header. The kurl is a custom header parameter providing the URL to the KMS to retrieve the DEK. The definition of the DCS container in JSON is provided in clause 8 of OGC 20-021r2. The responses from an OGC API Records request (assuming the GeoJSON conformance class) encrypted in a DCS container contain “metadata” (optional housekeeping information) and “data”, with “data” representing the encrypted original OGC API Records response.

The following Data Centric Security (DCS) flow applies for synchronous search (and subscription) requests.

- [1] The Catalog Client D114 (on the behalf of a signed in user) submits a request to the Catalog server (e.g. search request) and provides the DCS media type (f parameter), the access token, the key challenge (PIN), and the key challenge method.
- [2] The Catalog validates the access_token, applies access control based on the bearer token, and retrieves the username and client_id associated with the user token.
- [3] The Catalog server creates a Data Encryption Key (DEK) and then encrypts the response items using the relevant key (multiple keys might be required in case various confidentiality levels are associated to the response items).
- [4] The Catalog registers the DEK along the client id (aud parameter), the access token, and the key_challenge to Key Management Service D115 (KMS).
- [5] The Catalog response (DCS container) is returned to the Client.
- [6] The client extracts the DEK kid and kurl to request the DEK. Optionally, the client might provide the kek_id of a previously registered user public key (Key Encryption Key) to request the encryption of the key.
- [7] The Key Management Service (KMS) returns the DEK key optionally encrypted with the user private key (KEK).
- [8] The Catalog Client decrypts the DCS container and displays the records to the user.
6.2.1.2. Application/JOSE

Instead of the above approach where the catalog server chooses the encryption key and communicates the key to the client via a reference (kid) to the KMS, a client may choose to use its own public encryption key and pass that key to the catalog server. The KMS is not used in this scenario. When the response (JOSE format) is received, the client can decrypt the response with its own private (encryption) key.

See Annex A Example 3.2 for a detailed Jupyter Notebook example using the application/jose media type defined in [RFC 7515].

The current assumption is that the /collections/{collection-id}/items and /collections/{collection-id}/items/{record-id} responses are in-scope for the above encryption. The same approach can be applied to /collections and /collections/{collection-id} if appropriate, with encrypted JSON instead of GeoJSON representations. Clients can discover which resources (URL) accept requests for the application/dcs+geo or application/jose media types as this should be advertised by the server in the OpenAPI definition of the interface or even in an OpenSearch
OSDD document. This allows for a flexible use of such encrypted response capabilities by a (Catalog).

6.2.2. Asynchronous Communication with DCS

When the notification includes links to the subscribed resources, the nominal Data Centric Security process can be applied when the resources are retrieved by the subscriber depending on the catalog implementation.

In the case where the notification includes the records, participants explored the following models for encrypting the data.

- Using the KMS for retrieving the DEK generated by the catalog. However, Testbed participants were concerned about persisting user information on the catalog side and/or key permissions for the receiver endpoint which are not necessarily the same as the subscriber client.
- Providing a client public key to encrypt DEK generated by the catalog.
- Providing a DEK to the catalog generated by the client using the catalog private key.

Further recommendations are proposed to enforce DCS on the resource data when they are requested in the notification content.

### REQUIREMENT 20

<table>
<thead>
<tr>
<th>LABEL</th>
<th>/req/subscription/dcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT</td>
<td>If the subscriber requests resources in the notifications (HTTP Prefer header asserting a return=representation preference), then data centric security shall be applied (i.e., resources are encrypted) if the resource-uri specifies the dcs+geo format (f parameter).</td>
</tr>
<tr>
<td>A</td>
<td>The catalog shall encrypt the data using the DEK provided by the client and return the DEK identifier (if wrapped_dek property is provided).</td>
</tr>
<tr>
<td>B</td>
<td>The catalog shall generate a DEK to encrypt the data and return the DEK encrypted using the client public key (if public-key property is provided).</td>
</tr>
</tbody>
</table>

6.2.3. Record Insertions With DCS

According to clause 2.3.1 of the CFP, publishers shall be able to send new or updated content to the catalog server with encrypted content that fulfills DCS principles. This implies that the catalog server acts as a (DCS) client receiving DCS encrypted content and must implement the steps a client typically applies as described in OGC 21-020r1.
6.3. Proposed Interface Design

6.3.1. Synchronous Responses With DCS (Encryption)

The diagram below is the proposed resource definition diagram for an OGC API-Records implementation with support for DCS (encryption).

Figure 13 — Resource Definition Diagram for OGC API-Records With DCS (Synchronous Responses)
6.3.2. Asynchronous Responses With DCS

As mentioned earlier, when notifications are requested for including the records (with the return-representation preference), the content must be encrypted for applying Data Centric Security.

The following flow extends the OGC API-Records asynchronous flow with DCS.

- The Catalog Client (on the behalf of a signed user) submits a subscription request to the Catalog server (with the return-representation preference). The request includes either the public-key or wrapped-dek property (detailed further).
• The Catalog server registers the subscription and returns HTTP code 201 along the link to the created subscription.

• Each time the server prepares a notification, the records are encrypted depending on the selected security approach.

Figure 15 – DCS Async Sequence Diagram

6.3.2.1. Subscriber Submits a DEK

The subscriber might submit a DEK generated locally then used by the catalog to return the resources encrypted in the notifications. The notification content is encoded in a JWE token with a header mentioning the kid (client DEK identifier) for decrypting the notification payload. The major drawback of this method is the reuse of the same encryption key for all notifications, potentially for years.

In the request, the subscriber provides the wrapped-dek property which holds the JWE token that includes the encrypted JWK encoding of the client DEK. The client DEK is encrypted with the catalog public key advertised on /.well-known/jwks.json with the attribute use set to enc.

The following subscription flow applies to a notification encryption based on a DEK generated by the client.

• The catalog client (on the behalf of a signed user) retrieves the catalog server public key.

• The catalog client generates the DEK and encrypts the DEK using the catalog public key.

• The catalog client submits a subscription request to the catalog server and provides the encrypted DEK in the wrapped_dek property.

• The catalog server decrypts the DEK with the private key.

• For each notification, the catalog server encrypts the data records with the DEK.

• The receiver service of the catalog client decrypts the data with the DEK.
6.3.2.2. Subscriber Submits Public Keys

The subscriber might submit a public key used to return the DEK that will be created by the catalog. The subscriber provides the `public_key` property which consists of a JWK public key.

In the notifications, the resources are encoded in a JWE token which includes the DEK encoded with the public key.

The following subscription flow applies to a notification encrypted based on a DEK generated by the catalog server.

- The catalog client submits a subscription request to the Catalog server and provides a public key in the `public_key` property.
- For each notification, the catalog server generates a DEK and sends the encrypted notification along with the JWE token holding the DEK encrypted with the client public key.
- The receiver service of the catalog client decrypts the DEK with the private key, then decrypts the notification content with the DEK.
As the solution also has to be implemented using an HTTP GET operation, the recommendation below applies for the alternative operation.

**RECOMMENDATION 17**

When provided in an HTTP GET request, the public key used to return the DEK should be encoded using the deepObject style as per OpenAPI 3.0 - simple non-nested objects are serialized as paramName[prop1]=value1&paramName[prop2]=value2&...
**RECOMMENDATION 18**

**STATEMENT**

The server SHOULD only deliver notifications to an email delivery endpoint if a link containing a secret code and delivered to the target email address has been clicked by the receiver (to confirm the subscription).

---

**RECOMMENDATION 19**

**STATEMENT**

The server SHOULD only deliver notifications to an HTTP(s) delivery endpoint if a TBD.

---

### 6.3.3. Record Insertions With DCS

Publishers need to send new or updated content to the catalog server so that the interested parties can be notified as elaborated in the asynchronous communication related sections of this ER.

DCS principles might be applied to encrypt the content submitted on the catalog. This implies that the catalog server acts as a DCS client receiving DCS encrypted content, and is required to implement the steps a client typically applied as described in the Testbed-17: Data Centric Security ER (OGC 21-020r1).

For applying DCS to the OGC API-Records insert operation, Testbed participants proposed relying on the private key of the catalog. The public asynchronous encryption key can be advertised in `/./well-known/jwks.json`.

**RECOMMENDATION 20**

**STATEMENT**

The records inserted in a catalog should be encoded using the catalog public key provided in the `/./well-known/jwks.json` endpoint (with key entry provided with attribute `use` set with value `enc`).

The `cty` (content type) attribute of the Javascript Object Signing and Encryption (JOSE) header (i.e., first part of the JWE) should be set to the media type of the metadata record to be inserted in the catalog. For example, the value “geo+json” corresponds to “application/geo+json”, while “vnd.iso.19139+xml” corresponds to “application/vnd.iso.19139+xml”. This allows the receiving catalog to know what it should obtain after decryption.

The sequence required for applying DCS to a catalog record is illustrated on the diagram below and includes the following steps.

- The catalog client (publisher) retrieves the JSON Web Key Set (JWKS) containing the public key required for encrypting published content. The relevant key is the key item holding the ‘enc’ value (for encoding) in the `use` attribute.

- The catalog client encrypts the content with the catalog public key and submits the request to the catalog server.
• Typically, in an isolated secured runtime environment behind the catalog server, the submitted encrypted records are decrypted with the private key.

• Once the decryption is achieved, the record is added to the catalog database and the catalog client receives the confirmation response.

Figure 18 — Publisher with DCS sequence diagram

See Annex A Example 3.4 for a detailed Jupyter Notebook example using the POST operation and application/jose media type for publishing a metadata record.

The above sequence can be combined with asynchronous communication as illustrated in the diagram below.

• The catalog client (publisher) retrieves the JSON Web Key Set (JWKS) containing the public key required for encrypting published content. The relevant key is the key item holding the ‘enc’ value (for encoding) in the use attribute.

• The catalog client receives confirmation of the processing job handling the request.

• The catalog client encrypts the content with the catalog public key and submits the request to the catalog server.

• Typically, in an isolated secured runtime environment behind the catalog server, the submitted encrypted records are decrypted with the private key.

• Once the decryption is performed, the record is added to the catalog database and the callback endpoint submitted by the catalog client receives a notification.
6.3.4. Synchronous Responses With DCS (Signatures)

To address the integrity and authenticity requirements, metadata records or catalog responses can be encoded, returned, or made available in the JWS format [RFC 7515]. The structure of the JWS token (in Compact Serialization) is a special case of a JWT token [RFC 7519].

It contains three logical parts:

- JOSE Header
- JWS Payload
- JWS Signature

These are the main header properties:

**Table 5 — JWS header parameter names**

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>typ</td>
<td>Type — declares the IANA media type of this complete JWS. The values JOSE and JOSE+JSON are defined in the JWS specification and apply to both JWS and JWE tokens.</td>
<td>JWT [RFC 7519]</td>
</tr>
<tr>
<td>cty</td>
<td>Content Type — declares the IANA media type of the secured content (the payload).</td>
<td>JWT [RFC 7519]</td>
</tr>
</tbody>
</table>
See Annex A Example 3.5 for an example using the JWS (Compact Serialization) format. This example uses the public key included in the response to validate the signature.

To ensure that the public key is actually the expected sender’s, the signature can also be validated with the public key advertised by the sender as shown in Annex A Example 3.6.

JWS can also be serialized as JSON using the “JWS JSON Serialization”. This serialization can represent multiple signatures which is not possible with the “Compact Serialization”.

For use cases where the base64url encoding of the payload (metadata record or response) is not desired, the use of “JSON Web Signature (JWS) Unencoded Payload Option” [RFC 7797] can be considered. This option is to be further explored as future work.

A simpler approach “JWS Clear Text JSON Signature Option (JWS/CT)” [7] preserves the original JSON representation of a metadata record entirely and adds a signature key in the root of the JSON object with the value of the JWS (Compact Serialization) string (first and third part). Before computing the JWS string, the JSON object is made canonical using JCS [RFC 8785]). This option is attractive for static catalogs where metadata files are put as JSON files on cloud storage (S3). This permits the full readability of the original metadata record. This approach can for instance be applied to the following.

- OGC API-Records metadata records and crawlable catalogs
- OGC 17-003r2 metadata records
- STAC item metadata records

This option is only applicable to JSON-encoded metadata records (or responses) and will be further explored in future work. Signature creation and verification for JWS/CT is shown in Annex A Example 3.7 and Example 3.8.
SOFTWARE DESIGN
7.1. Software Static Architecture

The diagram below shows the original allocation of components to the different Testbed participants.

![Diagram of component allocation](image-url)

**Figure 20 — Component allocation**

As part of the Testbed-18 Secure, Asynchronous Catalogs task, the following client and server-side components were deployed and used for integration testing.

7.2. Interfaces Context

The figure below shows the interactions between the various components and the interfaces that were used.
7.2.1. I/F KMS — Interface with Key Management System

The interface with the Key Management System (KMS) is of particular importance as most components need to interact with the KMS. The interface is defined in detail in the corresponding Engineering Report OGC 22-014. The diagrams below give an overview of this RESTful interface in terms of its different resources.
Figure 22 — Resource Diagram (/dek)
Figure 23 — Resource Diagram (/dek/{key_id})
Figure 24 — Resource Diagram (/kek)

Figure 25 — Resource Diagram (/kek/{key_id})
7.3. Software Components Design

7.3.1. D112 — OGC CSW Catalog Server

Endpoint: https://cat.csiss.gmu.edu/ows18

Client Partner guide: https://cat.csiss.gmu.edu/CSW_Client-Guide.pdf

The OGC CSW Standard stipulates the interfaces for catalog services. The interfaces are divided into three categories: OGC service interface, CSW discovery interface, and CSW manager interface. Specifically, the GetCapabilities operation is the OGC service interface that provides summary information of the CSISS CSW catalog. The operations in the CSW discovery interface include GetRecords, GetRecordById, DescribeRecord, and GetDomain. The operations in the CSW manager interface include Transaction and Harvest. Asynchronous processing is supported for GetRecords and Harvest requests (via csw:ResponseHandler). The CSW server supports three modes of the Transaction operation (Insert, Update, Delete).

Unlike many other agencies that rely on commercial on-demand cloud infrastructure such as Amazon Web Service (AWS), the Center for Spatial Information Science and Systems (CSISS) developed a private cloud to provide services for EO data. The private cloud was developed using Apache Cloud stack and provides virtual machines to multiple operational services hosted by CSISS (including the CSISS CSW server in Testbed 18). Currently, there are more than 300 GPU cores, over 500GB RAM, and more than 500 TB storage, as well as a cluster of NVIDIA Tesla K80 GPUs on the server. The resources available continue to expand. To ensure the safety and reliability of operational services, all machines are hosted at the CSISS Data Center, George Mason University. Thanks to the Center’s state-of-the-art UPS systems, the data center can provide continuous power support during a power interruption. Also, the Center is monitored 24 hours a day, 7 days a week to prevent any potential risk.

Figure 26 — CSISS CSW Server System Architecture
7.3.2. D113 — OGC API-Records Catalog Server

Endpoint: https://ogc.demo.secure-dimensions.de/pycsw

Software: https://github.com/securedimensions/pycsw/tree/tb18

For Testbed 18, the OGC API-Records implementation provided by pyCSW (https://github.com/geopython/pycsw) was extended with the following specific functionalities.

- Publishing of records via HTTP POST and PUT with support for different content types: application/json, application/jose (JWE format).
- Requesting records in different DCS formats: application/jose (JWS and JWE), application/dcs+geo.

Figure 27 — D113 - OGC API-Records Catalog User Interface
• Support for HTTP header “Preference” to trigger asynchronous responses via the generic parameter “subscription.”

• The Catalog Service user interface and OpenAPI contain additional functionalities that allow the different DCS formats to be requested. The responses “links” section was extended with the DCS formats.

• The Catalog Service has implemented RFC 6750 behavior (Bearer access token use) when requesting DCS formats or when publishing a record.

• For development support, the additional functionality is available via OpenAPI.

One of the biggest challenges was how to extend an existing open source implementation (pyCSW) so that it supports DCS without changing the original code. This was nearly achieved. Most of the functionality was included in the flask wrapper using Python decorator and additional routes (POST and PUT). Only a few changes were actually required in the “ogc/api/records.py” file, mainly to support the rendering of additional links in the link section.

The implementation of the record publishing (HTTP POST and PUT) is considered a prototype for demonstration only. The implementation DOES NOT check the uploaded data for malicious / dangerous code. So, DO NOT USE IN PRODUCTION!

The actual overhead associated with producing JWE or JWS is very minimal, because a record is only a few kilobytes in size. Because of the expected record size as a few kB, all cryptographic operations take place in main memory. This approach is fine for processing a single record but already may consume noticeable main memory and produce processing latency when requesting a JWS or JWE for a collection of records. So, DO NOT USE IN PRODUCTION.

The catalog server is hosted on Gunicorn (Python WSGI server) behind NGINX on Ubuntu 22.04.

The catalog service implementation interacts with the Key Management Service to register symmetric encryption keys for producing DCS responses.

The catalog service implementation is used by the Subscription Management Service to execute triggers for created subscriptions.

7.3.3. D114 — Catalog Client, Subscriber, Publisher

The Catalog client is a web client built using React. The client uses the oidc-client package to handle user authentication with the Authenix server and store the all-important bearer token. Users can login with the button in the top right of the header if they require access to the Records server or Records subscriptions.

The client enables users to pull documents from both the OGC API-Records and CSW implementations using a variety of formats and security schemes. Since CSW only offers the ISO format unencrypted, this is the only format the client supports for CSW implementation instances. For OGC API-Records, the user can choose between JWE and XML formats. For the JWE format, they can specify either direct or key wrap encryption.
The metadata is visualized via a Leaflet map and a table which can be used to edit the Title of the metadata. Further work could be done to implement a more complete data editing/interaction system depending on the required use case, for example visualizing quicklook images, etc.

The JWE security functionality is enabled by the Jose NPM library. This library provides the functionality to handle the JWKs that the KMS returns when direct encryption is requested. It also deals with the asymmetric keys submitted when the user requests using key wrap. As these keys are currently hardcoded into the demo, a full implementation of this system would need to safely store those keys on a user's device. Keys could be generated per session, but when using this encryption method the user would not be allowed to logout and then return to the session.

The resource card row enables the user to view, store, and upload stored documents to either a CSW or an API-Records instance via the upload form.
The client also provides a form for creating subscriptions to the API-Records server. This currently only allows the user to subscribe to email notifications at a fixed interval, but could be extended further in future.

Figure 30 — React Client Upload Form

7.3.4. D115.1 — Authorization & Identity Management System

The Authorization and Identity Management System in OGC Testbed 18 is AUTHENIX, operated by Secure Dimensions.

7.3.5. D115.2 — Key Management Server

The Key Management Server in the OGC Testbed 18 is identical to the Testbed 17 implementation. The details of the KMS can be found in section 5.4 of the OGC 21-020r1
7.3.6. Jupyter Notebook Client

This client is implemented as a Jupyter Notebook. When run locally, such as using with VS-Code as depicted below, the Notebook interfaces with the Authorization & Identity Management System (D115) authenticating the user and retrieving the user’s access_token.

![Jupyter Notebook Client interaction with D115](image)

**Figure 32 — Jupyter Notebook Client interaction with D115**

In addition, the client demonstrates how ISO19139 Earth Observation metadata records describing series and dataset retrieved from the ESA FedEO server (EOVOC), can be inserted for use in the OGC API Server without modification. See also Example 3.4.
The Notebook can be run in the Google Colaboratory environment as well as it is depicted below.

The source code of the Jupyter Notebook is included as Annex A. The Notebook code acts as a client to many of the server components as shown below.
Figure 35 — Component Diagram Jupyter Notebook
FUTURE WORK
FUTURE WORK

8.1. Asynchronous Communication

8.1.1. Include Reference to Subscription Endpoint

Possibly use rel='monitor' link as defined by RFC 5989 to the /subscriptions/{subscription-id} endpoint which monitors the HTTP resource (i.e., search request) which produced the search result(s) in the search response (FeatureCollection).

Similarly, a subscription endpoint available for an OGC API-Records service should be discoverable in the landing page of the OGC API-Records service (in addition to being described in its OpenAPI definition) to allow clients to choose between the synchronous or the asynchronous interfaces.

8.1.2. OGC API-Notification Service

Propose Webhooks for an OGC API-Notification Service specification for other services besides Records which should work with the PubSub SWG.

8.1.3. Notification Content

The subscription prototype backend performs a periodic check as to whether the response corresponding to the resources-uri in the subscription object is identical or not to the original response. If not, then the same resources-uri is sent to the subscriber according to the delivery mechanism that was selected.

If the resources-uri corresponds to a single item (e.g., /items/{item-id}), then this means that the record content has changed. If the resources-uri corresponds to a search request (e.g., /items?...), then this means that the (first result page) of the response has changed.

A number of improvements are possible.

- In the case of a search request, changes should be detected that affect not only the first result page (e.g., first n results and information about total number of hits). A deletion followed by an insertion, keeping the total number of results identical, may not be detected.

- In case resources-uri corresponds to a search request, the client should have the option to receive the actual changes, instead of a repetition of the full set of results. This is particularly useful for very large collections or searches with many results. A mechanism to
receive information about deleted records is required in particular when the notification mechanism would be applied for supporting incremental harvesting. A mechanism similar to the Atom deleted-entry Element [RFC-6721] might be applied. The Open Archive Initiative Protocol for Metadata Harvesting [OAI-PMH], which is still widely used, has support for deleted records.

- When a resource-uri corresponds to a search result, e.g., “all records with datetime overlapping with date-1 and date-2”, it might be useful to indicate in a subscription that the dates are to be interpreted as relative with respect to the actual time the subscription check is executed, instead of interpreting dates always as absolute dates.

### 8.1.4. Asynchronous Patterns for OGC API Common

A significant topic for future activities should be exploring how asynchronous communications can be addressed at the OGC API Common level. In the context of other APIs, asynchronous communications must address a wider range of concerns than for OGC API Records, as shown in the following examples.

- A server might handle requests asynchronously for preparing the results (single response), or in response to certain events (multiple responses).
- A client should specify the preferred behavior for retrieving the delayed response(s) either actively (polling pattern) or passively (push pattern requiring a listener endpoint).

The next subsections propose patterns to address general OGC API concerns related to asynchronous communications.

- The asynchronous pattern for delayed response provides a simple solution to query (poll) a URL until the response is ready.
- The callback pattern for delayed response complements the asynchronous pattern for specifying an endpoint for receiving (push) the response(s).
- The OGC API Common Asynchronous requirement class is a draft that generalizes the API records asynchronous solution relying on jobs resources.

#### 8.1.4.1. Asynchronous Pattern for Delayed Response

Implementations of OGC APIs typically use a communication protocol pattern based on a single socket pair for sending and receiving an HTTP(S) request / response. The handling of slowly resolved queries that are not ready within the regular HTTP time out (i.e., roughly 30 seconds) can be addressed using a simple communication pattern (not implying the subscription resources solution detailed earlier).

The proposed standard RFC 7240 “Prefer Header for HTTP” recommends using the ‘Prefer’ header to indicate the server behavior preferred by the client. As already adopted by OGC API Processes, the “respond-async” preference indicates that the client prefers the server to respond
asynchronously to a response. In addition, the value `respond-async, wait=10` is a hint to the server that the client expects a maximum of 10 seconds to return the response following the traditional synchronous pattern.

RFC 7240 mentions that the server honoring the "respond-async" preference should return a 202 (Accepted) response as per HTTP 1.1 (RFC 7231). However, the actual behavior is unspecified, and little guidance is provided by code 202 ("The representation sent with this response ought to describe the request's current status and point to (or embed) a status monitor that can provide the user with an estimate of when the request will be fulfilled"). For the above reason, some Testbed participants also proposed considering the alternative code 303: See Others. "See Other" is a way to redirect web applications to a new URI, particularly after a HTTP POST has been performed.

Independent of the response code, the server should return a Location header (and an optional Retry-After header) holding a link to the target requested resources. The link to the target resources can monitor the status of the previous request until the resources are available. The structure of the monitor response is custom as it needs to fit the purpose. But, to ensure that the caller is not too proactive, the server may throttle the caller via 429 (too many requests: Retry-after).

![Asynchronous pattern for delayed response](image)

Figure 36 — Asynchronous Pattern for Delayed Response

### 8.1.4.2 Callback Pattern for Delayed Response

Complementary to the asynchronous pattern described above, the request might be extended to submit an endpoint URI for receiving callback messages that contain the delayed response of the server. Indeed, as adopted by OGC API Processes, OpenAPI 3.0 provides a callback (push-based)
mechanism where a subscriber-URL is passed to the API in the request. Once the resources are available, the result response is sent to the specified URL.

OpenAPI supports specifying the placeholders of the callback URI’s (potentially for a set of defined events) submitted in the request, and to define the schemas of the callback messages which must be specified in the callbacks property of the related definition of the OpenAPI operation. An example is shown below.

callbacks:
  completed:
    ’{$request.header.Prefer}/callbackURI’:
      post:    # Method
        requestBody: # Contents of the callback message
          ...
        responses: # Expected responses
          ...

Figure 37 — Generic OpenAPI Definition of the callback

For expressing the callback endpoint (and options), Testbed participants highlighted one simple approach taking advantage of the Prefer header. The header might be extended with a callback token holding the (single) endpoint for callback messages. Also, in case of multiple results (based on particular events), the frequency preference can be provided in a schedule token holding a unix-cron value.

8.1.4.3. OGC API Common Asynchronous Requirement Class

The proposed approach for a generic asynchronous requirement class relies upon the typical use of HTTP code 202. A job resource is created to monitor the execution of the request. Note that the approach is very similar and reuses most concepts from the OGC API Processes.

<table>
<thead>
<tr>
<th>RECOMMENDATION 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>
RECOMMENDATION 22

LABEL /req/async/response

A If a request is executed asynchronously, the server should respond with an HTTP status code of 202. The server should return a Location header (and an optional Retry-After header) holding a link to the job monitoring the processing of the request.

REQUIREMENT 21

LABEL /req/async/job

A The server shall support the HTTP GET operation for retrieving a long-running asynchronous job at the path /jobs/{jobID}.

B A successful execution of the operation shall be reported as a response with a HTTP status code 200. The content of that response shall be based upon the OpenAPI 3.0 schema jobStatus.yaml.

The jobStatus schema is illustrated on the class diagram below.

AsynchronousJob Class

A AsynchronousJob

id: String
status: String
start-time: Date
end-time: Date
request-uri: String
callback: String
schedule: String

results

A Result

href: String
creation-time: Date

Figure 38 — JobStatus Schema
### RECOMMENDATION 23

**LABEL** /req/async/prefer-callback

| A | If a request is accompanied with the HTTP Prefer header asserting a callback preference (endpoint URI), then the potential asynchronous response(s) should be pushed as a callback message delivered to the provided callback endpoint URI. |
| B | If a request is accompanied with the HTTP Prefer header asserting a callback preference and a schedule UNIX-cron value, then the potential asynchronous response(s) should be pushed in respect to the submitted schedule. |

The resulting sequence diagram is provided below.

**Figure 39 — Generic Asynchronous Job Sequence Diagram**

The status values defined in OGC API Processes are clarified below in the context of sequential results updates (on purpose) managed by the asynchronous job.

### REQUIREMENT 22

**LABEL** /req/async/job-status

| A | The status of a job shall be accepted if the asynchronous job request is valid and has been queued for execution. |
| B | The status of a job shall be running if the provided start time has been reached. |
| C | The status of a job shall be failed if the asynchronous job request is not valid or if the processing of the request raised an error that prevented its completion. |
8.2. Data Centric Security

8.2.1. Request for Specific Encrypted Media Type

The application/dcs+geo (JSON-based) media type as currently defined contains a data object and a metadata object. In the current document, the data object carries the encrypted payload, i.e., an encrypted catalog /items response (GeoJSON FeatureCollection) or /items/{item-id} response (GeoJSON Feature). The content of the metadata object is use-case specific and not required for decryption purposes.

While the current document assumes that the catalog responses (after decryption) are GeoJSON objects, this can be generalized to allow the encrypted payload to be used to carry the encrypted representation of other metadata formats including ISO19139, ISO19139-2, ISO19115-3, etc. The cty (content type) attribute in the JOSE header of the JWE already allows the specification of the media type (omitting “application/” when no additional “/” appears) that will result after decryption.

Similarly, it should be possible for a catalog client to request an application/dcs+geo or application/jose response and additionally specify the expected content-type (e.g., ../items?cty=vnd.iso.19139+xml) the client expects to obtain after decrypting the data object included in the application/dcs+geo or the application/jose response.

8.2.2. Generic Approach for Any OGC API

Future work should define how to apply DCS to all OGC APIs, with a JSON or XML-based container. Homogeneous solutions are preferred, allowing the use of XML or JSON encodings for all responses generated by the resources of the RESTful interface, avoiding the need to mix technologies and related tools.

8.2.3. DCS for XML via STANAG

While the above approach would allow for returning (and encrypting) XML-based responses with the JSON-based API, organizations may want to avoid mixing XML and JSON and may prefer having XML responses for all resources of the RESTful API, using an XML container (media
8.3. Authentication and Authorization

Future work should include migration from access tokens in Bearer HTTP header fields to access tokens in DPoP HTTP header fields demonstrating proof of possession (DPoP) as proposed in https://datatracker.ietf.org/doc/html/draft-ietf-oauth-dpop and RFC 7800.

GET /collections/datasets/items?... HTTP/1.1
Host: catalog.org
Accept: application/jose
Authorization: DPoP Kz~8mXK1EalYznwH-LC-1fBAo.4jp-zsPE_NeO gxU
DPoP: eyJ0eXAiOiJkcG9wK2p3dCI1mAmsZyI6IkVTR0FzWll0UmFzWll0UmFzWl0UmFzWll0 Um5zb3IsInRva2VuIjoid3d3dlczI6MjU2In0.

Figure 40 — Example Request With DPoP-Bound Access Token

8.4. DCS and Streaming

During previous OGC Testbeds, Data Centric Security was applied to different OGC APIs and Data Encoding Standards. The synchronous characteristics of OGC APIs introduce the requirement that the maximum latency of responses is limited by the socket timeout or by the acceptance to wait by the user. To keep the response waiting time short (before the first response byte is sent to the client) it is important that the data encoding can be streamed. Streaming a resource has the advantage that the first byte is more or less sent to the client immediately after the request has been received by the server. The full response is then transferred using chunk encoding. This technique reduces the burden on the server, as memory and disk space does not limit the total size of the data to be processed.

Work in previous testbeds determined that data encoding using the OGC GeoPackage Standard cannot be streamed as the underlying SQLite file format requires that the GeoPackage is processed on the server in full before the first byte is transferred to the client. However, there are other encodings that can be streamed: A GeoJSON feature collection, for example, can be produced and transferred using chunk encoding.

The research question should be addressed in the context of DCS in which geospatial data encoding is streaming-ready and can be used in combination with DCS. For example, is it possible to produce a JWS or JWE containing a feature collection in chunks to minimize the
burden on the server’s memory and disk space? Is it possible to create a STANAG 4774/4778 response in XML or JSON based on chunks? Is it possible to use a similar protocol like HLS (RFC 8216) to stream large volumes of encrypted data? Is it possible to support streams and byte ranges for encrypted data?

So a future task could be to study OGC Encoding Standards and determine their streaming capabilities and recommend their use with and without DCS applied based on prototype implementations.

8.5. DCS in Production

Both previous Testbeds and this one introduced DCS building blocks that extend the typical geodata infrastructure architectures with specific services: Authentication, Key Management, and the Subscription Service. The latter was introduced as a generic solution to handle synchronous to asynchronous communication bridging supporting WebPush and Web Callback including security and email.

To operate the architecture and APIs in a production environment, it is important to not only have covered the green flow of information. Software engineers also should consider the red flow: Which vulnerabilities exist in the protocols, APIs, and implementations that may enable an adversary to attack certain parts of the system.

For example, the use of (simple) Bearer access tokens on the Key Management System introduces the vulnerable to token replay attacks. An attacker, in possession of an active access token from another user could obtain all keys of that the user. Even worse, perhaps, delete existing keys. Other vulnerabilities may be introduced by implementations that support the user’s ability to upload data. In these cases, implementation vulnerabilities like SQL or XSS code injection may exist that can cause data or key breach.

The research question would be to determine what vulnerabilities exist in the current Testbed DCS architecture and in existing open-source implementations by applying security best practices from the IETF (e.g., 6819), W3C, and others. For example the security recommendation for using Bearer access tokens (draft RFC ‘OAuth 2.0 Security Best Current Practice’) introduce various attack vectors. Can these attack vectors be applied to the DCS architecture and what harm would they do if successfully executed?

Another research question is related to performance and scalability. The ultimate aspect for scalability is to keep the server side (upstream applications) stateless and to reduce burden on the resources to the minimum. Can that be achieved with the current DCS architecture? Which limitations and implications to scalability and performance exist?
8.6. Demonstrating Proof of Possession Access Token

The use of bearer access tokens is widely adopted but vulnerabilities to misuse stolen access tokens introduce a threat to DCS: Once a user has gotten an access token, it can be used with no further proof. So for example, a stolen active access token can be used for obtaining sensitive (encrypted) information and data encryption keys. This jeopardizes all efforts put into DCS.

The use of OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP) as defined in the IETF draft standard\(^1\) thwarts the successful use of stolen access tokens.

Future work should concentrate on studying the principles for using DPoP access (and refresh) tokens on a DCS architecture as outlined in Testbed 18. In particular, the use of DPoP access tokens bound to a resource URI or tagged for one time use should be studied and implemented with the DCS Server and the KMS to demonstrate the improvement over the use of bearer access tokens. This may require operating two infrastructures in parallel; one that operates bearer access tokens and one that operates DPoP access tokens. Successfully executed attacks with bearer access tokens should not be possible with the infrastructure that is based on DPoP.

8.7. My-(OGC) API

Leveraging DCS, it is possible to manage sensitive data (features) and metadata (records). Any publishing of records or features should be associated with a user. That user would have the authority over how to preserve continuous sensitivity / integrity. So, the end-to-end sensitivity / authenticity should be supported via OGC APIs.

Is it possible to introduce the concept of ownership for OGC APIs? Should the owner of a resource have the authority over the resource’s protection? Furthermore, queries for “my” resources and “Jane’s resources” would be possible. This would be an important step towards the use of OGC APIs in Citizen Science.

What would be required to apply the “my” characteristics to OGC APIs and, in particular, DCS extensions?

\(^1\)https://datatracker.ietf.org/doc/html/draft-ietf-oauth-dpop-11 at the time of writing
ANNEX A (INFORMATIVE) NOTEBOOK SECURE CATALOG
ANNEX A
(INFORMATIVE)
NOTEBOOK SECURE CATALOG

By Y. Coene, C. Noel (Spacebel s.a.), A. Matheus (Secure Dimensions GmbH), 21 October 2022

%pip install pyjwt
%pip install folium
%pip install jsonpath-ng
%pip install jcs

import json, requests, xml
from xml.dom import minidom
from xml.etree import ElementTree
from IPython.display import Markdown as md
import folium

# Verification of SSL certificate.
verify_ssl = True

endpoint_csw = 'https://cat.csiss.gmu.edu/ows18/csw'
endpoint_api_records = 'https://ogc.demo.secure-dimensions.de/pycsw'
endpoint_collection = endpoint_api_records + '/collections/metadata:main'
endpoint_items = endpoint_collection + '/items'
endpoint_queryables = endpoint_collection + '/queryables'

# URL to an item inserted in the catalog before.
endpoint_item = endpoint_items + '/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001'
endpoint_subscriptions = 'https://ogc.demo.secure-dimensions.de/sms/subscriptions'

def add_feature_to_map( m, fdata):
    # fdata is JSON dictionary representing a Feature
    # modifies the map 'm'

    key = 'geometry'
    if key in fdata:
        c = fdata[key]['coordinates']

        # create string with coordinates without separators
        polygon=str(c).translate(str.maketrans('', '', '[]'))
        list1 = polygon.split()

        points = []
        for i in range(0, len(list1),2):
            points = points + [(float(list1[i+1]), float(list1[i]))]

        folium.FitBounds(points, padding=(100,100)).add_to(m)
        folium.Polygon(points, popup=str(fdata), tooltip=fdata['id']).add_to(m)

    return
def show_on_map( data ):
    # data is JSON dictionary representing a FeatureCollection or Feature

center = [38.128, 2.588]
m = folium.Map(location=center, zoom_start=2)
folium.LayerControl().add_to(m)

key = 'geometry'
if key in data:
    add_feature_to_map(m, data)
    return m

# contains featurecollection or feature
key = 'features'
if key in data:
    n = len(data[key])
    if (len(data[key])<1): return m

key = 'geometry'
j=0
while (j<n):
    fdata = data['features'][j]  # use current feature of collection
    add_feature_to_map(m, fdata)
    j=j+1
return m

Figure A.1

Requirement already satisfied: pyjwt in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (2.4.0) [notice]
A new release of pip available: 22.1.2 -> 22.3 [notice] To update, run: python.exe -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: folium in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (0.12.1.post1)
Requirement already satisfied: jinja2>=2.9 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from folium) (3.1.2)
Requirement already satisfied: numpy in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from folium) (1.21.4)
Requirement already satisfied: branca>=0.3.0 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from folium)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from jinja2>=2.9->folium) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from requests->folium) (2020.6.20)
Requirement already satisfied: charset-normalizer~2.0.0 in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from requests-
folium) (2.0.7) [notice] A new release of pip available: 22.1.2 -> 22.3 [notice] To update, run: python.exe -m pip install --upgrade pip
Requirement already satisfied: jsonpath-ng in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (1.5.3) Requirement already satisfied: six in c:\users\yvesc\appdata\roaming\python\python38\site-packages (from jsonpath-ng) (1.15.0) Requirement already satisfied: ply in c:\users\yvesc\appdata\local\programs\python\python38\lib\site-packages (from jsonpath-ng) (3.11) Requirement already satisfied: decorator in c:\users\yvesc\appdata\roaming\python\python38\site-packages (from jsonpath-ng) (4.4.2) [notice] A new release of pip available: 22.1.2 -> 22.3 [notice] To update, run: python.exe -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.

Figure A.2

A.1. D115 (Identity Mgt System) authentication

if 'google.colab' in str(get_ipython()):
    # Running in Google Colab
    access_token = input("Enter access token obtained from https://ogc.demo.secure-dimensions.de/token-app/ :")
else:
    # Not running in Google Colab
    # Authenticate via Notebook from Secure Dimensions to obtain the "access_token".
    %run ./authenix.ipynb

Figure A.3

Random string of length 4 is: jghh Listening on port 8008 ...
GET */?code=9dfdd77a5e8936aea35c99995ec4918d43344e45&state=None HTTP/1.1 Host: 127.0.0.1:8008 Connection: keep-alive Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Sec-Fetch-Site: none Sec-Fetch-Mode: navigate Sec-Fetch-User: ?1 Sec-Fetch-Dest: document sec-ch-ua: "Chromium";v="106", "Google Chrome";v="106", "Not-A=Brand";v="99" sec-ch-ua-mobile: ?0 sec-ch-ua-platform: "Windows" Accept-Encoding: gzip, deflate, br Accept-Language: en-GB, en-US;q=0.9, en;q=0.8 Warning: state does not match! 9dfdd77a5e8936aea35c99995ec4918d43344e45

Figure A.4
A.1.1. D115 (Identity Mgt System) Get access token

```python
access_token
Figure A.5
'4e6e701e0eb6a7d13e76b33da136fa07a2c4d7fc'
Figure A.6
```

A.2. D113 (OGC API-Records) without DCS

A.2.1. D113 GET /items without DCS (GeoJSON)

Example: 1.1
> Access the OGC API-Records D113 /items without DCS (GeoJSON)

```python
response = requests.get(endpoint_items + '?limit=2&offset=8',
    verify=bool(verify_ssl),
    headers={'Accept': 'application/geo+json'})

data = json.loads(response.text)
jstr = json.dumps(data, indent=3)
md("```json

" + jstr + "```")
```

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "id": "9d075d6f-a85a-18fa-1f51-ced8-ef7c-5bce-a91eee54",
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        2.52, 50.64,
                        2.52, 51.51,
                        5.94, 51.51,
                        5.94, 50.64
                    ],
                    [
                        2.52, 50.64
                    ]
                ]
            }
        }
    ]
```

Figure A.7
"properties": {
  "externalId": "9d075d6f-a85a-18fa-1f51-ced8-ef7c-5bce-a91ee54",
  "datetime": "2022-07-01",
  "start_datetime": null,
  "end_datetime": null,
  "recordUpdated": "2022-07-21T22:40:13Z",
  "type": "service",
  "created": "2013-05-27",
  "updated": "2022-07-01",
  "title": "WMS Vlaamse Landmaatschappij",
  "description": "Web Map Service met data van de Vlaamse Landmaatschappij.",
  "keywords": [
    "OGC:WMS",
    "bodemeenheid",
    "bodemkaart",
    "gegeneraliseerde bodemkaart voor Vlaanderen",
    "infiltratiekenmerk",
    "kwel",
    "kwetsbaarheid",
    "landinrichting",
    "landschapsseenheid",
    "plattelandsbeleid",
    "ruimte\u00ebf",
    "ruimtelijke structuren in Vlaanderen",
    "inrichtingsproject",
    "landinrichtingsplan",
    "landinrichtingsproject",
    "onderzoek",
    "planprogramma",
    "vastgesteld",
    "afbakening",
    "planbegeleidingsgroep",
    "inrichtingsnota",
    "recht van voorkoop",
    "rvv",
    "voorkoopgebied",
    "voorkooprecht",
    "blokgrens",
    "kavelplan",
    "landeigendom",
    "ruilverkaveling",
    "ruilverkavelingsproject",
    "sectie",
    "gebruiksruil",
    "infrastructuurwerken",
    "natuurbehoud",
    "natuurinrichting",
    "natuurinrichtingsproject",
    "gebied met recht van voorkoop",
    "focusgebied",
    "grondwater",
    "mestdecreet",
    "mestactieplan",
    "nitraat"
  ]
}
"nitraatconcentratie",
"nitraatreisduw",
"nitraatreisduwdrempelwaarde",
"nitraatreisdustaal",
"oppervlaktewater",
"overschrijding",
"waterkwaliteit",
"beheerovereenkomst",
"agromilieu-klimaatmaatregel",
"ecosysteemdienst",
"plattelandsontwikkeling",
"landbouwbeleid",
"PDPO",
"GLB",
"agrobiodiveristeit",
"gebiedstype"
],
"associations": [
{
"name": "Capabilities van de webdienst WMS Vlaamse Landmaatschappij",
"description": null,
"type": "OGC:WMS",
"rel": "OGC:WMS"
},
{
"href": "https://geoservices.informatievlaanderen.be/raadpleegdiensten/VLM/wms",
"name": null,
"description": null,
"type": null,
"rel": null
},
{
"href": "https://geoservices.informatievlaanderen.be/raadpleegdiensten/VLM/wms",
"name": null,
"description": null,
"type": null,
"rel": null
},
{
"href": "https://geoservices.informatievlaanderen.be/raadpleegdiensten/VLM/wms",
"name": null,
"description": null,
"type": null,
"rel": null
}
],
"extent": {
"spatial": {
"bbox": [
2.52, 50.64, 5.94, 51.51
]
},
"crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5: \

- **MODE 1: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m**
MODE 2 WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
MODE 4 CHLOROPHYL BAND SET: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
MODE 5 LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

"associations": [
  {
    "href": "https://tpm-ds.eo.esa.int/oads/data/PROBA1-CHRIS/PR1_OPER_CHR_MO1_1P_20181005T134600_N51-350_E003-170_0001.SIP.ZIP",
    "name": "Download",
    "description": null,
    "type": "WWW:DOWNLOAD",
    "rel": "WWW:DOWNLOAD"
  },
  {
    "href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/browse/PR1_OPER_CHR_MO1_1P_20181005T134600_N51-350_E003-170_0001.SIP.ZIP_BID.PNG",
    "name": "QUICKLOOK",
    "description": null,
    "type": null,
    "rel": null
  },
  {
    "href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/thumbnail/PR1_OPER_CHR_MO1_1P_20181005T134600_N51-350_E003-170_0001.SIP.ZIP_TIMG.jpg",
    "name": "THUMBNAIL",
    "description": null,
    "type": null,
    "rel": null
  }
],
"extent": {
  "spatial": {
    "bbox": [
      3.07, 51.29, 3.28, 51.39
    ],
    "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
  }
},
"links": [
  {
    "rel": "via",
    "type": "application/xml",
    "title": "This document as XML",
    "href": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/PR1_OPER_CHR_MO1_1P_20181005T134600_N51-350_E003-170_0001?f=xml",
    "hreflang": "en-US"
  }
],
A.2.2. D113 GET /items without DCS (XML)

Example: 1.2

> Access the OGC API-Records D113 /items without DCS (XML) is not supported.

```python
response = requests.get(endpoint_items + '?f=xml',
    verify=bool(verify_ssl),
    headers={})
# headers={'Accept': 'application/xml'})
response.text
# xmlstr = minidom.parseString(response.text).toprettyxml(indent=' ',newl='')
# md(````xml
# + xmlstr + ```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
WFS Interessante plaatsen & Points of Interest

Points of Interest

Points of Interest

This document as XML

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ

Orthobeeldvorming - OMZ
Bundeling van de datasets die een overzicht geven van het groen in Vlaanderen voor bepaalde jaren.

boswijzer\&x22;\, &\&x22;landbouw\&x22;\, &\&x22;associaties\&x22;:

[\{&\&x22;href\&x22;: &\&x22;http://www.geopunt.be/download?container=groenkaart&title=Groenkaart%20Vlaanderen&\&x22;\, &\&x22;name\&x22;: &\&x22;Groenkaart%20Vlaanderen&\&x22;\, &\&x22;description\&x22;: \null, \&\&x22;type\&x22;: &\&x22;null\&x22;\, &\&x22;title\&x22;: \null\, &\&x22;This\document\as\XML\&x22;\, &\&x22;href\&x22;: &\&x22;http://www.opengis.net/def/crs/OGC/1.3/CRS84\&x22;\}]

CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5: MODE 1: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km MODE 2 WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km MODE 4 CHLOROPHYLL BAND SET: Full swath width, 18 spectral
bands, nadir ground sampling distance 17m @ 556km \u2022 MODE 5

LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **MODE 1**: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km
- **MODE 2**: WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 3**: thermodenudation bands
- **MODE 4**: Polarimetric bands
- **MODE 5**: LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **MODE 1**: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km
- **MODE 2**: WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 3**: thermodenudation bands
- **MODE 4**: Polarimetric bands
- **MODE 5**: LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
17m @ 556km MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km MODE 4 CHLOROPHYL BAND SET: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km MODE 5 LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **MODE 1**: 
- **MODE 3**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 4**: CHLOROPHYL BAND SET: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 5**: LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:
Full swath width, 62 spectral bands, 773nm\n / 1036nm, nadir ground\n sampling distance 34m @ 556km \u2022 MODE 2 WATER BANDS: Full\n swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km \u2022 MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling\n distance 17m @ 556km \u2022 MODE 4 CHLOROPHYL BAND SET: Full swath width, 18 spectral\n bands, nadir ground sampling distance 17m @ 556km \u2022 MODE 5 LAND CHANNELS: Half\n swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km All Proba-1 passes are systematically acquired according to the current acquisition\n plan, CHRIS data are processed every day to Level 1A and made available to ESA\n users. Observation over a new specific area can be performed by submitting the\n request to add a new site to the acquisition plan. &x22;; &x22;associations&x22;: [{&x22;href=&x22;: &x22;https://tpm-ds.eoa.esa.int/oads/data/PROBA1-CHRIS/\n PR1_OPER CHR_MO1 1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP&x22;; &x22;name=&x22;; &x22;Download&x22;; &x22;description&x22;: null, &x22;type=&x22;; &x22;WWW:DOWNLOAD&x22;; &x22;rel=&x22;; &x22;WWW:DOWNLOAD&x22;}, {&x22;href=&x22;: &x22;http://tpm-ds.eoa.esa.int/oads/meta/PROBA1-CHRIS/brow\n se/PR1_OPER CHR_MO1 1P_20220307T140200_N51-240_E002-920_0001.SIP.BID.PNG&x2\n 22;name=&x22;; &x22;QUICKLOOK&x22;; &x22;description&x22;: null, &x22;type=&x22;; &x22;null, &x22;rel=&x22;; null}], {&x22;href=&x22;: &x22;http://tpm-ds.eoa.esa.int/oads/meta/PROBA1-CHRIS/thumbnail/\n PR1_OPER CHR_MO1 1P_20220307T140200_N51-240_E002-920_0001.SIP.TIMP.JPG&x2\n 22;name=&x22;; &x22;THUMBNAIL&x22;; &x22;description&x22;: null, &x22;type=&x22;; &x22;null, &x22;rel=&x22;; null}], &x22;extent&x22;: [{&x22;spatial&x22;: [{&x22;bbox&x22;: [[2.83, 51.18, 3.02, 51.28]]}, &x22;crs&x22;: &x22;http://www.opengis.net/def/crs/O\n GC/1.3/CRS84&x22;}, &x22;}}], &x22;links&x22;: [{&x22;rel=&x22;; &x22;via&x22;; &x22;type=&x22;; &x22;application/xml&x22;; &x22;title&x22;; &x22;This document as XML&x22;; &x22;href&x22;; &x22;https://ogc.demo.secure-dimensions.de/pycs\n w/collections/metadata/main/items/PR1_OPER CHR_MO1 1P_20220307T140200_N51-240_E002-920_0001?\n f=xml&x22;; &x22;hreflang&x22;; &x22;en-US&x22;;], &x22;assets&x22;: []}, {&x22;id=&x22;; &x22;e17fe655-987c-4c5f-b6ae-b10dcd4fccc3&x22;; &x22;type=&x22;; &x22;Feature&x22;; &x22;coordinates&x22;: [[2.53, 50.67], [2.53, 51.51], [5.92, 51.51], [5.92, 50.67], [2.53, 50.67]]}, &x22;properties&x22;: [{&x22;externalId&x22;; &x22;e17fe655-987c-4c5f-b6ae-b10dcd4fccc3&x22;; &x22;datetime&x22;; &x22;2022-04-12&x22;; &x22;start_datetime&x22;; &x22;1997-01-01&x22;; &x22;end_datetime&x22;; &x22;2020-07-01&x22;; &x22;recordUpdated&x22;; &x22;2022-07-21T22:40:13Z&x22;; &x22;type&x22;; &x22;Biologische Waarderingekaarten en Natura 2000 Habitatkaarten - Toestand 2020&x22;; &x22;title&x22;; &x22;De Biologische Waarderingekaart (BWK) is een uniforme inventarisatie en evaluatie van het gehele Vlaamse grondgebied aan de hand van een set karteringseenheden die staan voor vegetatie, bodembedekking en kleine landschapselementen
(lijn- en puntvormige elementen). Ook met de aanwezigheid van
belangrijke fauna-elementen is er rekening gehouden. De vernieuwde
BWK, versie 2, probeert, in vergelijking met de versie 1, aan
meer vereisten en noden te voldoen, zowel inhoudelijk als op het
gebied van nauwkeurigheid. In deze versie van de BWK zijn ook de
Natura 2000 habitattypen opgenomen. In Vlaanderen komen actueel 47
Natura 2000 habitattypen van de Bijlage I van de Habitatrichtlijn
voor. Daarnaast zijn er in Vlaanderen ook 15 regionaal belangrijke
biotopen gedefinieerd. Dit zijn biotopen die naar biologische
waarden en belang voor de biodiversiteit vergelijkbaar zijn
met habitattypen, maar die op Europees niveau minder bedreigd
zijn. Deze kaart geeft de best beschikbare informatie anno
2020 over de verspreiding van de Natura 2000 habitattypen, de
regionaal belangrijke biotopen en de karteringseenheden van
De biologische Waarderingekaart. Dit kan een vereenvoudiging
zijn van de werkelijkheid op terrein. Ten allen tijde geldt
De kaart geeft de best beschikbare informatie anno
2020 over de verspreiding van de Natura 2000 habitattypen, de
regionaal belangrijke biotopen en de karteringseenheden van
De biologische Waarderingekaart. Dit kan een vereenvoudiging
zijn van de werkelijkheid op terrein. Ten allen tijde geldt

Kaartlabel BWK-karteringseenheden, Kaartlabel Habitattype, Kaartlabel Habitattype 3260, Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden

Kaartlabel BWK-karteringseenheden, Kaartlabel Habitattype, Kaartlabel Habitattype 3260, Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden

Kaartlabel BWK-karteringseenheden, Kaartlabel Habitattype, Kaartlabel Habitattype 3260, Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden

Kaartlabel BWK-karteringseenheden, Kaartlabel Habitattype, Kaartlabel Habitattype 3260, Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden, BWK 2 - Faunistisch belangrijke gebieden
null

null

null

null

null

null

null
CHRIS Level 1A acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **MODE 1**: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km
- **MODE 2 WATER BANDS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 3 LAND CHANNELS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 4 CHLOROPHYL BAND SET**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 5 LAND CHANNELS**: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
A.2.3. D113 GET /items/{itemId} without DCS (GeoJSON)

Example: 1.3

> Access the OGC API-Records D113 /items/{itemId} without DCS (GeoJSON)

```python
response = requests.get(endpoint_item ,
    verify=bool(verify_ssl),
    headers={'Accept': 'application/geo+json'})

data = json.loads(response.text)
jstr = json.dumps(data, indent=3)
md("```json

+ jstr +
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **MODE 1**: Full swath width, 62 spectral bands, 773nm/1036nm, nadir ground sampling distance 34m @ 556km
- **MODE 2**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 3**: Land channels, Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 4**: Chlorophyll band set, Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 5**: Land channels, Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
The ISO metadata record is available via the via link at JSONPath $.links[?(@.rel=='via')].

```python
from jsonpath_ng.ext import parse
expression = parse("$.links[?(@.rel=='via')].href")
r = expression.find(data)
r[0].value
```

Figure A.14

'https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001?f=xml'

Figure A.15

show_on_map(data)

Figure A.16

Make this Notebook Trusted to load map: File → Trust Notebook

### A.2.4. D113 GET /queryables without DCS (JSON)

**Example: 1.4**

> Access the OGC API-Records D113 /collections/{collection-id}/queryables without DCS to retrieve the available search parameters.

```python
response = requests.get(endpoint_queryables,
                         verify=bool(verify_ssl),
                         headers={'Accept': 'application/json'})
```

```python
data = json.loads(response.text)
jstr = json.dumps(data, indent=3)
md("```json

```

```

```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
A.2.5. D113 GET /items/{itemId} without DCS (XML)

Example: 1.5
> Access the OGC API-Records D113 /items/{itemId} without DCS (XML) to retrieve an ISO19139 metadata record.

response = requests.get(endpoint_item + '?f=xml',
    verify=bool(verify_ssl),
    headers={})
# headers={'Accept': 'application/xml'})
xmlstr = minidom.parseString(response.text).toprettyxml(indent='    ',newl='')
md("```xml
    " + xmlstr + "```")

Figure A.19

```xml
<?xml version="1.0" ?>
<gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
    xmlns:gco="http://www.isotc211.org/2005/gco"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:gmx="http://www.isotc211.org/2005/gmx"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.isotc211.org/2005/gmd ./apiso-inspire.xsd">
  <gmd:fileIdentifier>
    <gco:CharacterString>PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001</gco:CharacterString>
  </gmd:fileIdentifier>
  <gmd:language>
  </gmd:language>
  <gmd:parentIdentifier>
  </gmd:parentIdentifier>
  <gmd:hierarchyLevel>
    <gmd:MD_ScopeCode codeList="http://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_19139_Schemas/resources/codelist/ML_gmxCodelist.xml#MD_ScopeCode" codeListValue="dataset"/>
  </gmd:hierarchyLevel>
  <gmd:contact xmlns:gmd="http://www.isotc211.org/2005/gmd">
    <gmd:CI_ResponsibleParty>
      <gmd:organisationName>
        <gco:CharacterString>ESA/ESRIN</gco:CharacterString>
      </gmd:organisationName>
      <gmd:positionName>
        <gco:CharacterString>ESRIN Earth Observation Help Desk</gco:CharacterString>
      </gmd:positionName>
    </gmd:CI_ResponsibleParty>
    <gmd:contactInfo>
      <gmd:CI_Telephone>
        <gmd:voice>
          <gco:CharacterString>+3906941801</gco:CharacterString>
        </gmd:voice>
        <gmd:facsimile>
          <gco:CharacterString>+390694180280</gco:CharacterString>
        </gmd:facsimile>
      </gmd:CI_Telephone>
    </gmd:contactInfo>
  </gmd:contact>
</gmd:MD_Metadata>
CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5: • MODE 1: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km • MODE 2 WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km • MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km • MODE 4 CHLOROPHYL BAND SET: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km • MODE 5 LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km • All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
<gmd:CI_ResponsibleParty>
  <gmd:organisationName>
    <gco:CharacterString>ESA/ESRIN</gco:CharacterString>
  </gmd:organisationName>
  <gmd:positionName>
    <gco:CharacterString>Earth Observation helpdesk</gco:CharacterString>
  </gmd:positionName>
  <gmd:contactInfo>
    <gmd:CI_Contact>
      <gmd:phone>
        <gmd:CI_Telephone>
          <gmd:voice>
            <gco:CharacterString>+3906941801</gco:CharacterString>
          </gmd:voice>
        </gmd:CI_Telephone>
      </gmd:phone>
      <gmd:address>
        <gmd:CI_Address>
          <gmd:deliveryPoint>
            <gco:CharacterString>Largo Galileo Galilei 1</gco:CharacterString>
          </gmd:deliveryPoint>
          <gmd:city>
            <gco:CharacterString>Frascati (Roma)</gco:CharacterString>
          </gmd:city>
          <gmd:postalCode>
            <gco:CharacterString>00044</gco:CharacterString>
          </gmd:postalCode>
          <gmd:country>
            <gco:CharacterString>Italy</gco:CharacterString>
          </gmd:country>
          <gmd:electronicMailAddress>
            <gco:CharacterString>eohelp@esa.int</gco:CharacterString>
          </gmd:electronicMailAddress>
        </gmd:CI_Address>
      </gmd:address>
    </gmd:CI_Contact>
  </gmd:contactInfo>
  <gmd:role>
    <gmd:CI_Contact>
      <gmd:phone>
        <gmd:CI_Telephone>
          <gmd:voice>
            <gco:CharacterString>+390694180280</gco:CharacterString>
          </gmd:voice>
        </gmd:CI_Telephone>
      </gmd:phone>
      <gmd:address>
        <gmd:CI_Address>
          <gmd:deliveryPoint>
            <gco:CharacterString>Largo Galileo Galilei 1</gco:CharacterString>
          </gmd:deliveryPoint>
          <gmd:city>
            <gco:CharacterString>Frascati (Roma)</gco:CharacterString>
          </gmd:city>
          <gmd:postalCode>
            <gco:CharacterString>00044</gco:CharacterString>
          </gmd:postalCode>
          <gmd:country>
            <gco:CharacterString>Italy</gco:CharacterString>
          </gmd:country>
          <gmd:electronicMailAddress>
            <gco:CharacterString>eohelp@esa.int</gco:CharacterString>
          </gmd:electronicMailAddress>
        </gmd:CI_Address>
      </gmd:address>
    </gmd:CI_Contact>
  </gmd:role>
  <gmd:CI_Contact>
    <gmd:phone>
      <gmd:CI_Telephone>
        <gmd:voice>
          <gco:CharacterString>+3906941801</gco:CharacterString>
        </gmd:voice>
      </gmd:CI_Telephone>
    </gmd:phone>
    <gmd:address>
      <gmd:CI_Address>
        <gmd:deliveryPoint>
          <gco:CharacterString>Largo Galileo Galilei 1</gco:CharacterString>
        </gmd:deliveryPoint>
        <gmd:city>
          <gco:CharacterString>Frascati (Roma)</gco:CharacterString>
        </gmd:city>
        <gmd:postalCode>
          <gco:CharacterString>00044</gco:CharacterString>
        </gmd:postalCode>
        <gmd:country>
          <gco:CharacterString>Italy</gco:CharacterString>
        </gmd:country>
        <gmd:electronicMailAddress>
          <gco:CharacterString>eohelp@esa.int</gco:CharacterString>
        </gmd:electronicMailAddress>
      </gmd:CI_Address>
    </gmd:address>
  </gmd:CI_Contact>
  <gmd:onlineResource>
    <gmd:CI_OnlineResource>
      <gmd:linkage>
        <gmd:URL>http://www.esa.int</gmd:URL>
      </gmd:linkage>
    </gmd:CI_OnlineResource>
  </gmd:onlineResource>
</gmd:CI_ResponsibleParty>
<gmd:CI_RoleCode>
  <gmd:roleValue>originator</gmd:roleValue>
</gmd:CI_RoleCode>
</gmd:CI_ResponsibleParty>
</gmd:pointOfContact>
</gmd:graphicOverview>
<gmd:descriptiveKeywords>
  <gmd:MD_Keywords>
    <gmd:keyword>
    </gmd:keyword>
    <gmd:keyword>
    </gmd:keyword>
    <gmd:keyword>
      <gmx:Anchor xlink:href="https://gcmd.earthdata.nasa.gov/kms/concept/c7b5c02c-724d-4a19-b824-98180f3900c9">EARTH SCIENCE</gmx:Anchor>
    </gmd:keyword>
    <gmd:keyword>
      <gmx:Anchor xlink:href="https://gcmd.earthdata.nasa.gov/kms/concept/5debb283-51e4-435e-b2a2-e8e2a977220d">EARTH SCIENCE</gmx:Anchor>
    </gmd:keyword>
  </gmd:MD_Keywords>
</gmd:descriptiveKeywords>
</gmd:MD_BrowseGraphic>
</gmd:graphicOverview>
</gmd:CI_ResponsibleParty>
</gmd:CI_RoleCode>
<gmd:date>
</gmd:date>

</gmd:CI_Citation>

</gmd:thesaurusName>

</gmd:descriptiveKeywords>

</gmd:MD_LegalConstraints>

</gmd:otherConstraints>


</gmd:accessConstraints>

</gmd:otherConstraints>


</gmd:otherConstraints>

</gmd:MD_LegalConstraints>

</gmd:resourceConstraints>

</gmd:MD_LegalConstraints>

</gmd:resourceConstraints>

</gmd:spatialRepresentationType>

</gmd:language>


</gmd:temporalElement>

</gmd:EX_TemporalExtent>

</gml:TimePeriod>

<gml:beginPosition>2022-03-07T14:02:00Z</gml:beginPosition>

<gml:endPosition>2022-03-07T14:06:00Z</gml:endPosition>
<gmd:name>
</gmd:name>

<gmd:description>
</gmd:description>

<gmd:function>
</gmd:function>

<gmd:CI_OnlineResource>
  <gmd:onLine>
    <gmd:MD_DigitalTransferOptions>
      <gmd:MD_Distribution>
        <gmd:distributionInfo>
          <gmd:DQ_DataQuality>
            <gmd:scope>
              <gmd:DQ_Scope>
                <gmd:level>
                  <gmd:MD_ScopeCode codeList="http://standards.iso.org/iso/19139/resources/gmxCodelists.xml#MD_ScopeCode" codeListValue="dataset"/>
                </gmd:level>
                <gmd:levelDescription>
                  <gmd:MD_ScopeDescription>
                    <gmd:other>
                      <gco:CharacterString>Dataset</gco:CharacterString>
                    </gmd:other>
                  </gmd:MD_ScopeDescription>
                </gmd:levelDescription>
              </gmd:DQ_Scope>
              <gmd:DQ_DomainConsistency>
                <gmd:result>
                  <gmd:DQ_ConformanceResult>
                    <gmd:specification>
                      <gmd:CI_Citation>
                        <gmd:title>
                        </gmd:title>
                      </gmd:CI_Citation>
                      <gmd:date>
                        <gmd:CI_Date>
                          <gmd:date>
                            <gco:Date>2010-12-08</gco:Date>
                          </gmd:date>
                          <gmd:dateType>
                            <gco:DateType/>
                          </gmd:dateType>
                        </gmd:CI_Date>
                      </gmd:date>
                    </gmd:specification>
                  </gmd:DQ_ConformanceResult>
                </gmd:result>
              </gmd:DQ_DomainConsistency>
            </gmd:scope>
          </gmd:DQ_DataQuality>
        </gmd:distributionInfo>
      </gmd:MD_Distribution>
    </gmd:MD_DigitalTransferOptions>
  </gmd:onLine>
</gmd:CI_OnlineResource>
A.3. D113 GET /items with DCS (KMS)

Example: 2.1
> Access the OGC API-Records D113 /items with DCS (application/dcs+geo)

# Get the access token from https://ogc.demo.secure-dimensions.de/token-app/
# access_token = '4572e42da2c093012606449ba29da41fad2d599b'
# response = requests.get(endpoint_items + '?key_challenge=123&limit=1&access_token='+access_token,
response = requests.get(endpoint_items + '?key_challenge=123&limit=1&type=series',
    verify=bool(verify_ssl),
    headers={'Accept': 'application/dcs+geo', 'Authorization': 'Bearer ' + access_token})
response
A.3.1. JWT

Catalogue response corresponds to a JWT. The complete response is a JWE. The syntax is defined at https://openid.net/specs/draft-jones-json-web-encryption-02.html. Contains 3 parts separated by dots...

token = response.text

token

Figure A.22

Figure A.23
A.3.2. Decode the JWT

See https://stackoverflow.com/questions/64616462/decode-jwt-header-python

test = json.loads(token)
obj = test['objects']

Figure A.24
encdata = obj[0]['data']

Figure A.25

ejhgbCiiqjklaXiIcLjcm0d0jpbIntcmwiXXSwiY3R5iojyXBBwblGjYXRpb2b4zV2Vv2kpb2b4
O6ahKoMPMrzJsoIwKvFh8x4Rgh7T-3GLglrBXXwJ5qznQDXGw3uzKgMHB0AVB78z12jJFmLcw7Ps
qbWVeKKnBaFcmIOTT1QWsA41p34nc7h8jwY9779CU2PiB9bmq8aobDp_6JXz8RieJgmcK0ziT
TDQ1pB8kzKVFvmOhdL9KnWwqLg8Lvuldijr9T61anauWE4cLyIg6A4qDQhV2Fcgw5653yf
H1hIoBByfbysCG3J8h8e4nbnfDBGcVlH-UCENOCz981luXbP57sscXFMb9mww2BR-E
KJRGbzg-k_p00jkVIO-4T1izi0j0OGTdBHPmgQswjy56C-
L_RYBFmF7MUBGuvT3kZyQyVR2VZBavM72WFEFagRt6P7JgsqoDQC__ODkXJyufZBipaBLipj35
JS9XOBVt94pse5j2Po1sBteo2zDfJem9UReNwmoScrgTme478VqErCHBoN03KggwuOWP6InnwQuC1y
wknbwYo-
NIsn1fFueHmNjrjebPWUQfbu3UKMLV7U7Qe6XustZEvEDN_Ny5pzn5pCjuM1Cnp3eOoKwnbR
RnXLTa0QyxfxEVABeX4L42zF0c57RLK0sbljQE3qsCjAz7L7TWTB8eSV6EF033pPC_D0k2zgeLe9A-
_mkXrKkodTnagiYVEN9Nt_-MGmyR2xSwvskjB5iqi82FTyLbHg0xc-
Ht2HRGn7QeMYR4LK-KzPv2Ctsu-3-xcmwMMHRA0RqAecGt-
t8PIRXSy8emsqFcd4_euh5M6cHLYdipdKdMtqroKq06M-
tiYsBwTcosec6_zapy7LIGbP5E38Pj4mAs8kJwiWFDESCIGbLTDamy3i5OwWkGGKqCq2lxgdrl-
VRS84aj4jqiHe3GqCxhYyF1vE05v0sSbb9z2Kv3s0iwTa33D6j1zGk1ZzztyGNh6hfqFCRCaAzdApqy6p-
ipwTnaYeW0FqS_MRLgj1Zor3dBKEsb2cq7dPzXVwJndfXL8WDDKI1tdoHMeCdo_j_Lw7AXGjIg8CS-
vULCddupGillUV5PCgC1rY2o-
entMgqMrUGBwholKbNsc1cbdbc72jWFBatKPjV0Z_yJAvOs51pDGdwnaIDK9Lj1I5nCOCsQ5-
coBlUuUw3wbNm2rddUWEje7yHML9keqie0gUhnmOdfW93p1K8o8ojWBcrNUItBXTnHgf1efNh2BvCg-
bub5-TfL-
iwQKn9vrxURyGyroTZZRXIhhm8PhzThg43nCLFik6dGOG270oqsIm5d_sGsl7WagAc6aiEnrtIco-
moG1yWNEn1ZsrelijpeaBisRmzTO2rXnJ0DOYUN_Xgd5TtdgxTpgpgr_WR8u0mU8U_WBYthu-
YHL-Q9Zzd0CDfNjhl-
bgoHez7BSxXwDu0P1qjnKjFL6qAfTWBceE2XJ0M1r7G2CxbSdqDXKxeFrz4G08MTS4_Vmwy5c8h5p-
wktQwtBDqggVFKFxog717Gj0aWOWs8B6iwQ333xaKBYxZLF3zW6xo_4u0edXBM5xhRuQkhZ-
GMbTwFkvz0DSWyLEOKT-
FOgTNQpeE9CH342ZzXwOXLKehQmoSmeJELKJymW3GFt7iczfysNa4jn20_3nIHGoyGP3ctf0Ae-
pluCh40iklRJR4z5G5G0cs5KZg-k5xtcdVDDTVd4PVSu80vU8ggV0-
JNQwx73y_bnuMKULmgPpRhMrMRT-
Wnv3Ck7uWvn7jmSfLqTauB85F8EUyC8tuTcgs5ph5TzsDIRaco17dngcVxxtNgsYFWS0u-
yWmnmq9ehHcmili1L6sCMvrlLPudhr2rCpccxy1U0ngV0VrPugSaA8-
FdvUUNsYbceqFzT9goXFtaWrPgk8y4dDpeFbEWFt9sFe1inrzMd5Q8btvMKrE3y9ioUJXyGo-
PVMKlJ00h5RuchQ90x-
-Lyt2_BLOFKaqISzyCRZ6eVMhpaPvt09jApBcwZU2H6L0rzu-7hDhsoOih50-
zECrm5w6c18p9aMjsFhww9onBNNF3K3LTrugqWy-
-DEIlvKjIby3vonHvoso8q8smCpsH6EY_YFavrFMHyK6rQNYKACG37dbhiSkpizibtay3Npm-
uh4af5qiFvnmxUchZPPHS_HhrVAXBkgKwX_0k8fscF7dKvKrcR3x8U6Ddu1L0Dm60lkNh8s-
DSD9yn60OOGdD0l1LhsTX19lt7nplqQMMylfL8q_a7a-BDJ9kq-
DrOi6eWmGwprhuvbmdxvHfp1JxyEr_27K5G5p-FpaQwca5k5sQlTBsuep0-
p946a5v9Lz23ul6wyinjX7TeEkLMQVUZdUHeqNLY27-
ffpXKgLp4aY03o2NnB6BqHooEy7ulxpsfY3PDaMLxMw1HRHIFk-
FirmSiWng13Nj7kxpkyac6Zyfhi0B5p19IKWdfULPKpF0aqhKEJBD0_GDN67GnMqMnsFS0wUJ32P-
hKbU8I80hNNZeGy-
Bw2VvmG53a2GZp9xyfLPoDoW80o1J6UPWYY8p75hc176w5iXozufu9ezQW5ucxOz0I5CA77b12-
w0466BzTdbmW8fZDWAUP6zW-
import json
from jwcrypto.common import base64url_decode

parts = encdata.split('.

# get the first part of the token
jsondata = base64url_decode(parts[0])
data = json.loads(jsondata)
data

kid = data['kid']
kid

'1b506e2f-f576-4a9a-9570-5f406991cdb2'

# kurl is a custom field, used for the testbed.
kurl = data['kurl']
kurl

'https://ogc.demo.secure-dimensions.de/kms/dek/1b506e2f-f576-4a9a-9570-5f406991cdb2'

A.3.3. Get the key from KMS via de kid

See https://ogc.demo.secure-dimensions.de/kms/developers

# response = requests.get("https://ogc.demo.secure-dimensions.de/kms/dek/" + kid,
response = requests.get( kurl,
    verify=bool(verify_ssl),
    headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})
data = json.loads(response.text)
data
{'kid': '1b506e2f-f576-4a9a-9570-5f406991cdb2', 'alg': 'A256GCM', 'kty': 'oct', 'k': 'nPOHB8puwtwdAnqA4a5pb-iKOp9Rg077vdgUvGpyrOU', 'issuer': 'https://ogc.demo.secure-dimensions.de/pycsw', 'expires': 1666366239, 'issued_at': 1666365939, 'aud': '4f9610ac-9d1f-3797-6a41-6b18f9852572', 'sub': '4e7f5c63-0552-30f5-8b56-03e634ecd978'}

Figure A.34

A.3.4. Decode the payload

```python
import jwt
from jwcrypto import jwk, jwe
from jwcrypto.common import JWSEHeaderParameter

# create the DEK from the response obtained from the KMS.
dek = jwk.JWK.from_json(response.text)

registry = {
    'kurl': JWSEHeaderParameter(description='Key URL', mustprotect=True, supported=True, check_fn=None)
}

jwe_token = jwe.JWE( header_registry = registry )
jwe_token.deserialize(encdata)
jwe_token.decrypt(dek)

decrypted_payload = jwe_token.payload

data = json.loads(decrypted_payload)
jstr = json.dumps(data, indent=3)
md(f"```json

{json.dumps(data, indent=3)}```

```

Figure A.35

```json
{
    "type": "FeatureCollection",
    "features": [
    {
        "id": "05f9e795-b1ab-4ff7-9d5b-96c8c6d19d99",
        "type": "Feature",
        "geometry": {
            "type": "Polygon",
            "coordinates": [
            [
                2.53, 50.67,
            ],
            [
                2.53, 51.51,
            ],
            [
                5.92, 51.51,
            ],
            [
                5.92,
            ]
```
A.4. D113 GET /items/{item-id} with DCS (KMS)

Example: 3.1
> Access the OGC API-Records D113 /items/{item-id} with DCS (application/dcs+geo)

```python
# Get the access token from https://ogc.demo.secure-dimensions.de/token-app/
response = requests.get(endpoint_item + '?key_challenge=123',
    verify=bool(verify_ssl),
    headers={'Accept': 'application/dcs+geo', 'Authorization': 'Bearer ' + access_token})
```

```
<Response [200]>
```

A.4.1. JWT

Catalogue response corresponds to a JWT. The complete response is a JWE. The syntax is defined at https://openid.net/specs/draft-jones-json-web-encryption-02.html Contains 3 parts separated by dots...

token = response.text

token

```
{"type": "dcs+geo", "timestamp": "2022-10-21T17:25:42+02:00", "objects": [{"metadata": {"confidentiality_information": {"policy_identifier": "TB18", "classification": "unclassified"}, "creation_data_time": "2022-10-21T17:25:42+02:00", "number_matched": 1, "number_returned": 1, "bbox": {"bbox": [[2.83, 51.18, 3.02, 51.28]], "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"}, "links": [{"rel": "self", "type": "application/dcs+geo", "uri": 
```
A.4.2. Decode the JWT

See https://stackoverflow.com/questions/64616462/decode-jwt-header-python

test = json.loads(token)
obj = test['objects']
encdata = obj[0]['data']

cFigure A.41

A.4.2. Decode the JWT

See https://stackoverflow.com/questions/64616462/decode-jwt-header-python

test = json.loads(token)
obj = test['objects']
encdata = obj[0]['data']

cFigure A.42
import json
from jwcrypto.common import base64url_decode

parts = encdata.split('.

# get the first part of the token
jsondata = base64url_decode(parts[0])
data = json.loads(jsondata)
data

Figure A.43

{'alg': 'dir', 'crit': ['kurl'], 'cty': 'application/geo+json', 'enc': 'A256GCM', 'kid': '6a9e5727-34c6-4045-a42d-7125cb366d2a', 'kurl': 'https://ogc.demo.secure-dimensions.de/kms/dek/6a9e5727-34c6-4045-a42d-7125cb366d2a'}

Figure A.44

# header contains the 'kid' which can be requested to the KMS.
kid = data['kid']
kid

Figure A.46

'6a9e5727-34c6-4045-a42d-7125cb366d2a'

Figure A.47

# header contains the 'kurl' which is the URL to retrieve the key with 'kid' from the KMS.
kurl = data['kurl']
kurl

Figure A.48

'https://ogc.demo.secure-dimensions.de/kms/dek/6a9e5727-34c6-4045-a42d-7125cb366d2a'

Figure A.49
A.4.3. Get the key from KMS via de `kid`

See https://ogc.demo.secure-dimensions.de/kms/developers

```python
response = requests.get( kurl,
    verify=bool(verify_ssl),
    headers={
        'Accept': 'application/json',
        'Authorization': 'Bearer ' +
        access_token})

data = json.loads(response.text)
data
```

Figure A.50

```json
{'kid': '6a9e5727-34c6-4045-a42d-7125cb366d2a', 'alg': 'A256GCM',
'kty': 'oct', 'k': 'UbFiy-2i3nzPkaUVEvkDF29FD0N7vRUjN5ACBvoohA0',
'issuer': 'https://ogc.demo.secure-dimensions.de/ pycsw', 'expires': 1666366243, 'issued_at': 1666365942,
'aud': '4f9610ac-9d1f-3797-6a41-6b18f9852572', 'sub':
'4e7f5c63-0552-30f5-8b56-03e634ecd978'}
```

Figure A.51

A.4.4. Decode the payload

```python
import jwt
from jwcrypto import jwk, jwe

# create the DEK from the response obtained from the KMS.
dek = jwk.JWK.from_json(response.text)

registry = {
    'kurl': JWSEHeaderParameter(description='Key URL',
    mustprotect=True,
    supported=True, check_fn=None)
}

jwe_token = jwe.JWE(header_registry = registry )
jwe_token.deserialize(encdata)
jwe_token.decrypt(dek)
decrypted_payload = jwe_token.payload

data = json.loads(decrypted_payload)
jstr = json.dumps(data, indent=3)

md("```json

```

```
jstr + "
```json
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **2022 MODE 1**: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km
- **2022 MODE 2 WATER BANDS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **2022 MODE 3 LAND CHANNELS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **2022 MODE 4 CHLOROPHYL BAND SET**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **2022 MODE 5 LAND CHANNELS**: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
A.5. D113 GET /items/{item-id} with DCS (JOSE)

Example: 3.2
> Access the OGC API-Records D113 /items/{item-id} with DCS (application/jose) and public key. See https://www.rfc-editor.org/rfc/rfc7515.html.

# Prepare client key pair for encryption/decryption (public and private)

cli_kid = "clikid"
ce_key = jwk.JWK(generate='RSA', kid=cli_kid, use="enc" )
ce_key.export(False, True)
# Convert the public key into a comma-separated string to pass as query parameter (as per OpenAPI example)

c_{pk} = ce_key.export(private_key=False, as_dict=True)

cli_public_key = ",
".join([key, str(value)])

for key, value in ce_{pk}.items()

cli_public_key

response = requests.get(endpoint_item + '?public_key=' + cli_public_key,

verify=bool(verify_ssl),

headers={'Accept': 'application/jose', 'Authorization': 'Bearer ' + access_token})

response

A.5.1. Get JOSE response

Catalogue response is an application/jose structure (RFC7515) encoded as a series of base64url-encoded values (some of which may be the empty string), each separated from the next by a single period (\'.\') character. Next steps are the same as the application/dcs+geo decoding, but the decryption key is known by the client and does not need to be retrieved from the KMS with information obtained in the response.
encdata = response.text
encdata
import json
from jwcrypto.common import base64url_decode

parts = encdata.split('.
')
jsondata = base64url_decode(parts[0])
data = json.loads(jsondata)
data

{'alg': 'RSA-OAEP', 'cty': 'application/geo+json', 'enc': 'A128GCM',
'kid': 'clikid'}

# private key needed to decrypt the content
dek_str = ce_key.export(private_key=True, as_dict=False)
dek = jwk.JWK.from_json(dek_str)

A.5.2. Decode the payload

Use as `dek` the private key of the client key pair used for encryption/decryption (ce_key)
dek.export()

Figure A.65

'"d":"C15sP_l9yqYvvO-
kyeIPMFHHeifHUaWkgs3o2pcBdoUUGbDBFqcn7XXcFaWHSkdoHp6ta1oKDj3JMxXFeFQlLTyKcK1YmZBa_w-aNZ8-
E15awJ7nIFn0cOCOKUERmoF2tscN1_0r1JYNY7UkEA3ssOoyj0DmDK5sMlbmmBeUTSeuvA885C5
GD4eXAM-24RqITOBX6Xmp5S191IOw9QAbHtElLCGAFRT6QkPAY7u3TlzJNYacQDUUMbnd-
Jd_85wh9V0pw8eyvhyjCzniNz8buUKPbV0OjkkI2X8FG8c","dq":"2N6O8ZqomUpjVbI78NUeSGA0ZMcni98Q7d0gJsg809gLbujLa4a4210es1rTBlogGrD_6UFrdzJPm_3e9F9U","e":"AQAB",

oswjMGrhgSRDaj-n4B_anrQ6z0hrz58aC0yEI_ueNzndpOEFCoA5vhqjungHv9ggqa1vIQq4GDbEsjYA5iJ828UQvPTqT9B0Ib4TW2HeCcVoBxr0h4kiSwLFreEFSE2QjCih4tScMFzZdY985s2sDf0qxdU7c4h84J_y292pB6TfGWgltAHfpmkwrC1qiQUqcMrw8X896eIIYjUjQdKnVc5nfwa0cUy8KLC29m0zAVWuuzUJHsnZ BuJNdn1mUnPla1K4msbDGj5TAZjLMWC4riq13YYoYidizp5QuGibu43hUmJUtGrabEi0Jlkowo xFigVECy30ZEtJX-zndpVw-
HTbtRpc","q":"8G8P6XNxl_gnFx5H7jVAEx0bZu0jwrj6VxkQDM49BFdH8JdxTAYn6urhYIeJeGf7B8QM","qi":"McWfM-
FqXLIw7t8Q8okcLc8K6iKvk94XwCj909SrZydrEnDbOgpj8CHcElkdALTc-T-j6ckPwMb9Is0TjkjUavQz_WNSyaS1HUE394DjYawb1lz042PbczfQTGkoA6mmT-ScOb8ToN01mUJ1qEj-J-HkLvqrMy3r9YryZCs","use":"enc}'

Figure A.66

Decompress the payload with thedek (private key of the client)

jwe_token = jwe.JWE()
jwe_token.deserialze(encdata)
jwe_token.decrypt(dek)

decrypted_payload = jwe_token.payload

data = json.loads(decrypted_payload)
jstr = json.dumps(data, indent=3)

```json

```

# jwe_token.header

Figure A.67

{
  "id": "PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001",
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        2.83, 51.18,
        2.83, 51.28,
        3.02, 51.28,
      ]
    ]
  }
}
CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

- **2022 MODE 1**: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km
- **2022 MODE 2 WATER BANDS**: Full swath width, 18 spectral bands, nadir sampling distance 17m @ 556km
- **2022 MODE 3 LAND CHANNELS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **2022 MODE 4 CHLOROPHYL BAND SET**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **2022 MODE 5 LAND CHANNELS**: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.
A.6. D113 POST /items without DCS

Example: 3.3
> Insert records to OGC API-Records D113 /items without DCS

```
# Get an ISO19139 (dataset) record from FedEO (EOVOC) to insert in the OGC API-Records server.
)

# xmlstr = minidom.parseString(response.text).toprettyxml(indent=' ',newl='')
# md("``\nxml\n`` + xmlstr + \"\n``
response = requests.post(endpoint_items, data = response.text,
```
```
verify=bool(verify_ssl), 
headers={'Accept': 'application/json', 'Content-Type': 'application/xml', 
'Authorization': 'Bearer ' + access_token})

response.headers

Figure A.69

{'Server': 'nginx/1.23.1', 'Date': 'Fri, 21 Oct 2022 15:25:49 GMT', 'Content-Type': 'application/json', 'Content-Length': '2', 
'Connection': 'keep-alive', 'Host': 'ogc.demo.secure-dimensions.de', 
'User-Agent': 'python-requests/2.26.0', 'Accept-Encoding': 'gzip, deflate', 
'Accept': 'application/json', 'Authorization': 'Bearer 4e6e701e0eb6a7d13e76b33da136fa07a2c4d7fc', 
'Location': 'https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/KO2_OTPF_KO2_MSC_2F_20091107T041750_20091107T041750_017498_E082_N028'}

Figure A.70

# Get an ISO19139 (service) record from FedEO (EOVOC) to insert in the OGC API-Records server.

headers={'Accept': 'application/xml'})

# xmlstr = minidom.parseString(response.text).toprettyxml(indent=' ',newl='')
# md("```xml
" + xmlstr + "```")

response = requests.post(endpoint_items, data = response.text, 
verify=bool(verify_ssl), 
headers={'Accept': 'application/json', 'Content-Type': 'application/xml', 
'Authorization': 'Bearer ' + access_token})

response.headers

Figure A.71

{'Server': 'nginx/1.23.1', 'Date': 'Fri, 21 Oct 2022 15:25:49 GMT', 'Content-Type': 'application/json', 'Content-Length': '2', 
'Connection': 'keep-alive', 'Host': 'ogc.demo.secure-dimensions.de', 
'User-Agent': 'python-requests/2.26.0', 'Accept-Encoding': 'gzip, deflate', 
'Accept': 'application/json', 'Authorization': 'Bearer 4e6e701e0eb6a7d13e76b33da136fa07a2c4d7fc', 
'Location': 'https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/eo-pdgs-landsat-datacube'}

Figure A.72

# Delete the record that was just inserted. 204 response means deletion was successful.

response = requests.delete(endpoint_items + '/KO2_OTPF_KO2_MSC_2F_20091107T041750_20091107T041750_017498_E082_N028', 
verify=bool(verify_ssl), 
headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})
A.7. D113 POST /items with DCS (JOSE)

Example: 3.4

> Insert records to OGC API-Records D113 /items with DCS (application/jose) and public key.

As a first step, obtain the public key of the Catalog via https://ogc.demo.secure-dimensions.de/.well-known/jwks.json.

```python
response = requests.get('https://ogc.demo.secure-dimensions.de/.well-known/jwks.json', verify=True, headers={'Accept': 'application/json'})
response.text
```

```python
import jwt
from jwcrypto import jwk, jwe

# Get the keyset from the catalog server (receiver)
server_keyset = jwk.JWKSet.from_json(response.text)
server_keyset
```

```json
{"keys": [{"kid": "catalog-sign-key", "kty": "RSA", "n": "lo0cM6QC6jUB6wnMM75S5px20PkLBuycsA6qqJoFwPkJXyFFx5Z6C6jNTq6MF_Dzje0EgI7EnILU2E6JN83Dz_sJDEdrC5qNjEDLMnLJTzsrnOcdD3FrXiDTolXjmEjnjmAfjL3TvQ", "e": "AQAB", "use": "sig"}, {"kid": "catalog-wrap-key", "kty": "RSA", "n": "utfDC-oXhPJV8n2KY- Lus_H6zn_LxsK6UNHySSXaAeu1c1I5QUM782pvWI1iSxq0s6a1a2faGocfr5X5P7ugmQaTRHVT NzaC1QF5IEDMVHC9HgytpJcmCAev0vHR800e7hSncJwbaa_Gryh3th6Y7PcmL-seGCJHPrLIMVeTaSr-65gV050VzsV4Fw0bq_1AdObq1I1F1SkHsyay0kIeC4hqO9uY5WqDkhA HeAE3M2UunIyKLFdXghntlb9UKJavi_", "e": "AQAB", "use": "enc", "key_ops": "wrapKey", "kid": "893ef3c8-c249-47a2-91e2-001a0b201647"}, {"kid": "catalog-sign-key", "thumbprint": "QmOPE5GEOX6TAP101446..."]}
```

```python
import jwt
from jwcrypto import jwk, jwe

# Get the keyset from the catalog server (receiver)
server_keyset = jwk.JWKSet.from_json(response.text)
server_keyset
```

```json
{"keys": [{"kid": "catalog-wrap-key", "thumbprint": "c14HuddqPaJdyPyf7HBe4qR98eabtEIyptGnQlo4s\"}, {"kid": "catalog-sign-key", "thumbprint": "
```
se_key_kid = 'catalog-wrap-key'
# Get the keypair for encryption by the catalog from the keyset via its 'kid'
# as there may be multiple.
se_key = server_keyset.get_key(kid=se_key_kid)
# From the pair, get the public key.
se_key.export(private_key=False, as_dict=True)

payload = response.text

# 'cty' tells receiver what content-type to expect after decryption (i.e. the
# type of the secured content). "application/" is left out of the media type.
# 'kid' key hint references the public key to which the JWE was encrypted;
# this can be used to determine the private key needed to decrypt the JWE.
# See https://www.rfc-editor.org/rfc/rfc7516

protected_header = {'alg': 'RSA-OAEP',
                    'cty': 'xml',
                    'enc': 'A128GCM',
                    'kid': se_key_kid}

# Encrypt the payload with the dek (public key of the server)
jwe_token = jwe.JWE(plaintext = payload.encode('utf-8'), recipient = se_key,
                    protected = protected_header)
enc = jwe_token.serialize(compact=True)  

Figure A.81

# Insert the metadata record (JOSE) in the catalog with 'POST'.
response = requests.post(endpoint_items, data = enc,  
    verify=bool(verify_ssl),
    headers={'Accept': 'application/json', 'Content-Type': 'application/jose',  
     'Authorization': 'Bearer ' + access_token})

response  

Figure A.82

<Response [201]>

Figure A.83

response.headers['Location']

Figure A.84

'https://ogc.demo.secure-dimensions.de/pycsw/collections/  
metadata:main/items/  
SP4_OPER_HRI__I__2A_20030714T104451_20030714T104500_000213_0042_0246'  

Figure A.85

Retrieve the metadata record that was inserted from the OGC API-Records catalog.

# extract location of new image from HTTP header  
url_location = endpoint_api_records + response.headers['Location']

# get it unencrypted.  
response = requests.get(url_location, verify=bool(verify_ssl), headers={  
    'Accept': 'application/geo+json'})
data = json.loads(response.text)  
show_on_map(data)

Figure A.86

Make this Notebook Trusted to load map: File → Trust Notebook

A.8. D113 GET /items/{item-id} with DCS (JWS)

---

Example: 3.5  
> Access the OGC API-Records D113 /items/{item-id} to get a signed response (JWS format). The signature is used to verify the integrity of the response. The media type and value of the `f` parameter are subject to change but allow to demonstrate the response format. The response is returned in clear (part 2 of the JWS token).

response = requests.get(endpoint_item + '?f=jws',  
    verify=bool(verify_ssl),
    headers={'Accept': 'application/jose;profile=jws'})
The JWS response consists of three parts separated by a `.` as explained in https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3

```
encdata = response.text
parts = encdata.split('.')
```

# JWS token part 1 contains information about the key to use for verifying the signature.
```
from jwcrypto.common import base64url_decode
header = base64url_decode(parts[0])
```

```
b'\{"alg":"RS256","cty":"geo+json","iss":"https://ogc.demo.secure-dimensions.de","jku":"https://ogc.demo.secure-dimensions.de/.well-known/jwks.json","jwk": \{"alg":"RS256","e":"AQAB","kid":"catalog-sign-key","kty":"RSA","n":"lo0cM6QC6jUB6wnMM7S5px2OPkLbuycs6qqJoFWPkXyFFxF5Z6CJNOc7li4WBprIAX65Sh6aOq3bBZ8FP3v7R7mzo7e5fckvMT...uLcc_ZtZt3pRBkgA_ZHZ7LJawhnkMGq_hpSMmDxkDoUHsJ5PLH1yATPhquQwdUWajUZJMs4S7Ns0LgCDoj0zeixq5K6lip4Mh6hSoRep8aB4UwshdV2VUbpBP-Tq6MF_DzjeOlgIEnILu2E6JN83Dz_sjDEdrC5qNjEDLMnLJTzsrmOcdD3FrX1DTolXjmEjnq-mAfjL3IvQ\},"kid":"catalog-sign-key","typ":"JWT"\}'
```

# The "jwk" property contains the public key to verify the signature
```
from jwcrypto import jwk
data = json.loads(header)
```

```
\{"alg": 'RS256', 'cty': 'geo+json', 'iss': 'https://ogc.demo.secure-dimensions.de', 'jku': 'https://ogc.demo.secure-dimensions.de/.well-known/jwks.json', 'jwk': \{"alg": 'RS256', 'e': 'AQAB', 'kid': 'catalog-sign-key', 'kty': 'RSA', 'n': 'lo0cM6QC6jUB6wnMM7S5px2OPkLbuycsA6qqJoFWPkXyFFxF5Z6CJNOc-7li4WBprIAX65Sh6aOq3bBZ8FP3v7R7mZOo7e5fckvMTPOAzizkjXQuN8ZZoQta6x...uLcc_ZtZt3pRBkgA_ZHZ7LJawhnkMGq_hpSMmDxkDoUHsJ5PLH1yATPhquQwdUWajUZJMs4S7Ns0LgCDoj0zeixq5K6lip4Mh6hSoRep8aB4UwshdV2VUbpBP-Tq6MF_DzjeOlgIEnILu2E6JN83Dz_sjDEdrC5qNjEDLMnLJTzsrmOcdD3FrX1DTolXjmEjnq-mAfjL3IvQ\}, 'kid': 'catalog-sign-key', 'typ': 'JWT'\}
```

# extract the embedded public key needed to verify the signature
```
key = jwk.JWK.from_json(json.dumps(data['jwk']))
```

```
OPENGEOGRAPHICCONSORTIUM22-018
```
Figure A.95

```python
# JWS token part 2 (the actual response)
from jwcrypto.common import base64url_decode
payload = base64url_decode(parts[1])
```

Figure A.96

```
b'"id": 
"PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001",
"type": "Feature", "geometry": {
"type": "Polygon", "coordinates": 

```
| 2.83, 51.18 |
```
```
| 2.83, 51.28 |
```
```
| 3.02, 51.28 |
```
```
| 3.02, 51.18 |
```
```
| 2.83, 51.18 |
```
```
"externalId": 
"PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001",
"datetime": "2022-06-30T14:16:24", "end_datetime": "2022-03-07T14:00:00Z",
"location": "2022-07-21T22:40:13Z", "type": "dataset",
"Proba CHRIS Level 1A", "description": "CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:

**MODE 1**: Full swath width, 62 spectral bands, 773nm/1036nm, nadir ground sampling distance 34m @ 556km

**MODE 2 WATER BANDS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km

**MODE 3 LAND CHANNELS**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km

**MODE 4 CHLOROPHYL BAND SET**: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

```
"associations": 

```
| "href": "https://tpm-ds.eo.esa.int/oads/data/PROBA1-CHRIS/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP",
"name": "Download", "description": null, "type": "WWW:DOWNLOAD", "rel": "WWW:DOWNLOAD"}
```
```
| "href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/browse/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP_BID.PNG",
"name": "QUICKLOOK", "description": null, "type": null, "rel": null}]
```
```
| "href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/thumbail/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP_TIMG.jpg",
"name": "THUMBNAIL", "description": null, "type": null, "rel": null}
```
```
"extent": 
{"spatial": 
{"bbox": [[2.83, 51.18, 3.02, 51.28]], "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"]}
```
```
```
Verify the signature and thus the integrity of the response with the key extracted from part 1 of the header. If an exception is raised, then the signature does not match.

```
from jwcrypto import jws
jwstoken = jws.JWS()
```

Figure A.99
jwstoken.deserialize(raw_jws = response.text)
# Verify the signature with the key from the header
jwstoken.verify(key)
payload = jwstoken.payload

```
b'{"id": "PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001", "type": "Feature", "geometry": {"type": "Polygon", "coordinates": [[[2.83, 51.18], [2.83, 51.28], [3.02, 51.28], [3.02, 51.18], [2.83, 51.18]]], "properties": {"externalId": "PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001", "datetime": "2022-06-30T14:16:24", "start_datetime": "2022-03-07T14:02:00Z", "end_datetime": "2022-03-07T14:06:00Z", "recordUpdated": "2022-07-21T22:40:13Z", "type": "dataset", "created": "2019-05-22", "updated": "2022-06-30T14:16:24", "title": "Proba CHRIS Level 1A", "description": "CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5: 

\u2022 MODE 1: Full swath width, 62 spectral bands, 773nm / 1036nm, nadir ground sampling distance 34m @ 556km 

\u2022 MODE 2 WATER BANDS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km 

\u2022 MODE 3 LAND CHANNELS: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km 

\u2022 MODE 4 CHLOROPHYL BAND SET: Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km 

\u2022 MODE 5 LAND CHANNELS: Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km 

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan.

"associations": ["href": "https://tpm-ds.eo.esa.int/oads/data/PROBA1-CHRIS/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP", "name": "Download", "description": null, "type": "WWW:DOWNLOAD", "rel": "WWW:DOWNLOAD"], ["href": "https://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/browse/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP_BID.PNG", "name": "QUICKLOOK", "description": null, "type": null, "rel": null], ["href": "https://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/thumbnail/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP.ZIP_TIMG.jpg", "name": "THUMBNAIL", "description": null, "type": null, "rel": null], ["spatial": {"bbox": [[2.83, 51.18, 3.02, 51.28]], "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"}], ["application/dcs+geo", "title": "This document as DCS + GeoJSON", "href": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata/main/items/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001?
```
```
Example: 3.6

> The signature is used to verify the integrity of the response, however, the key is retrieved from the originator of the response instead of using the included public key.

```python
response = requests.get('https://ogc.demo.secure-dimensions.de/.well-known/jwks.json', verify=True, headers={'Accept': 'application/json'})
response.text
```

---

Figure A.101

Figure A.102

Figure A.103

At the end of the request response, the response text is printed, which includes the keys obtained from the catalog server's well-known location. These keys are used for signing and verifying responses.
from jwcrypto import jwk, jwe

# Get the keyset from the catalog server (receiver)
server_keyset = jwk.JWKSet.from_json(response.text)
server_keyset

Figure A.104

{"keys": ["{"kid":"catalog-wrap-key","thumbprint":"cqHuddqPaJd1vpyf7HBe_qR98eabtEIypuFgnQl04s "}","{"kid":"catalog-sign-key","thumbprint":"uwatboW3aEXg3z0g1C16yQzze5ZbpvYC_abhbTnwXs8"}",{""kid ":"893ef3c8-c249-47a2-91e2-001a0b201647",""thumbprint":"ygIxjy7LMfiUVHdWEExEd5p3tv3SnJgkF8yT4B6M"}"]}

Figure A.105

# JWS header contains the "kid" of the key to be used for signature check.
data['kid']

'catalog-sign-key'

Figure A.106

# JWS header contains also the well-known URL we just used to retrieve the server's keys.
data['jku']

'https://ogc.demo.secure-dimensions.de/.well-known/jwks.json'

Figure A.107

Figure A.108

se_key_kid = data['kid']

# Get the keypair for encryption by the catalog from the keyset via its 'kid' as there may be multiple.
key = server_keyset.get_key(kid=se_key_kid)

# Display the public key.
key.export(private_key=False, as_dict=True)

Figure A.109

Figure A.110


Figure A.111

from jwcrypto import jws

# decrypt the same JWS with the key directly retrieved from the server.
jwstoken = jws.JWS()
jwstoken.deserialize(raw_jws = encdata)

# Verify the signature with the key obtained from the well-known URL at the server.
jwstoken.verify(key)
Figure A.112

payload = jwstoken.payload

payload

b'"id": "PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001",
"type": "Feature",
"geometry": {
  "type": "Polygon",
  "coordinates": [[[2.83, 51.18], [2.83, 51.28], [3.02, 51.28], [3.02, 51.18], [2.83, 51.18]]],
},
"properties": {
  "externalId": "PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001",
  "datetime": "2022-06-30T14:16:24",
  "start_datetime": "2022-03-07T14:02:00Z",
  "end_datetime": "2022-03-07T14:06:00Z",
  "recordUpdated": "2019-05-22",
  "created": "2019-05-22",
  "updated": "2022-06-30T14:16:24",
  "title": "Proba CHRIS Level 1A",
  "description": "CHRIS acquires a set of up to five images of each target during each acquisition sequence, these images are acquired when Proba-1 is pointing at distinct angles with respect to the target. CHRIS Level 1A products (supplied in HDF data files, version 4.1r3) include five formal CHRIS imaging modes, classified as modes 1 to 5:
- **MODE 1:** Full swath width, 62 spectral bands, 773nm/1036nm, nadir ground sampling distance 34m @ 556km
- **MODE 2 WATER BANDS:** Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 3 LAND CHANNELS:** Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 4 CHLOROPHYL BAND SET:** Full swath width, 18 spectral bands, nadir ground sampling distance 17m @ 556km
- **MODE 5 LAND CHANNELS:** Half swath width, 37 spectral bands, nadir ground sampling distance 17m @ 556km

All Proba-1 passes are systematically acquired according to the current acquisition plan, CHRIS data are processed every day to Level 1A and made available to ESA users. Observation over a new specific area can be performed by submitting the request to add a new site to the acquisition plan."

"associations": [
  {"href": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001?f=dcs+geo"},
  {"href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/browse/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP_BID.png"},
  {"href": "http://tpm-ds.eo.esa.int/oads/meta/PROBA1-CHRIS/thumbnail/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001.SIP_ZIP_TIMG.jpg"},
],
"extent": {
  "spatial": {
    "bbox": [2.83, 51.18, 3.02, 51.28],
  },
  "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"},
"links": [{"rel": "alternate", "type": "application/dcs+geo", "title": "This document as DCS + GeoJSON"},
"href": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items/PR1_OPER_CHR_MO1_1P_20220307T140200_N51-240_E002-920_0001?f=dcs+geo", "hreflang": "en-US"},
]}
A.9. Other signatures with DCS (JWS/CT)

Example: 3.7

> JWS/CT (Clear Text) signature creation: The signature is included in the original JSON metadata record (payload). See https://www.ietf.org/archive/id/draft-jordan-jws-ct-08.txt.

# Get a JSON metadata record (OGC 17-003r2) to be used for demonstrating the encoding.

```python
response = requests.get('https://eovoc.spacebel.be/collections/datasets/items/ASA_APS_1PNE2A00211115_100404_0000152011_00151_03712_0000?mode=owc', verify=True,
headers={'Accept': 'application/geo+json'})
datastr = response.text
data = json.loads(datastr)
```

Figure A.114

# create a key for signing (private+public).
```
srv_kid = "srvkid"
ss_key = jwk.JWK(generate='RSA', kid=srv_kid, use="sig")
```

Figure A.115
```python
import jwt
import jcs
from jwcrypto import jwk, jwe, jws

protected_header = {
    'alg': 'RS256',
    'kty': "RSA",
    'cty': 'geo+json',
    'kid': 'srv_kid'
}

# normalise the JSON to compute the JWS.
payload = str(jcs.canonicalize(data))

jwstoken = jws.JWS(payload.encode('utf-8'))
jwstoken.add_signature(key = ss_key,
    alg = None,
    protected = protected_header)

enc = jwstoken.serialize(compact = True)

# Remove middle part of the token to create a JWS token without payload.
parts = enc.split('.')
jwsstr = parts[0] + '...'.join(parts[2])

Create the JWS/CT

# add signature property to the original JSON object (data) and print result.
data['signature'] = jwsstr

jstr = json.dumps(data, indent=3)
md("```json
" + jstr + "```")
```
```
"geometry": {
  "coordinates": [
    [
      6.17, 55.89
    ],
    [
      5.8, 55.01
    ],
    [
      7.45, 54.78
    ],
    [
      7.85, 55.66
    ]
  ],
  "type": "Polygon"
},
"id": "https://evoc.spacebel.be/collections/datasets/items/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000?mode=owc",
"type": "Feature",
"properties": {
  "identifier": "ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000",
  "kind": "http://purl.org/dc/dcmitype/Dataset",
  "parentIdentifier": "ENVISAT.ASA.APS_1P",
  "productInformation": {
    "referenceSystemIdentifier": "epsg:4326",
    "productType": "ASA_APS_1P",
    "availabilityTime": "2002-11-15T10:04:19.914Z"
  },
  "links": {
    "data": [
      {
        "href": "https://esar-ds.eo.esa.int/oads/data/ASA_APS_1P/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000.N1",
        "type": "application/x-binary",
        "title": "Download"
      }
    ],
    "previews": [
      {
        "href": "http://esar-ds.eo.esa.int/oads/meta/ASA_APS_1P/browse/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000.N1_BID.JPG",
        "type": "image/jpeg",
        "title": "QUICKLOOK"
      },
      {
        "href": "http://esar-ds.eo.esa.int/oads/meta/ASA_APS_1P/thumbnail/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000.N1_TIMG.JPG",
        "type": "image/jpeg",
        "title": "THUMBNAIL"
      }
    ]
  }
}
"title": "OGC 17-084r1 metadata",
"type": "application/geo+json"
],
"alternates": [
{
"type": "application/atom+xml",
"title": "Atom format"
},
{
"type": "application/gml+xml;profile="http://www.opengis.net/spec/EOMPOM/1.1"
"title": "OGC 10-157r4 metadata"
},
{
"href": "https://eovoc.spacebel.be/collections/datasets/items/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000?httpAccept=application/gml+xml;profile=om10"
"type": "application/gml+xml;profile=om10"
"title": "OGC 10-157r3 metadata"
},
{
"href": "https://eovoc.spacebel.be/collections/datasets/items/ASA_APS_1PNESA20021115_100404_000000152011_00151_03712_0000?httpAccept=application/geo+json;profile="http://www.opengis.net/spec/ogcapi-features-1/1.0"
"type": "application/geo+json;profile="http://www.opengis.net/spec/ogcapi-features-1/1.0"
"title": "OGC 17-069r3 metadata"
},
{
"type": "application/geo+json;profile=https://stacspec.org"
"title": "STAC metadata"
},
{
"type": "application/vnd.iso.19139+xml"
"title": "ISO 19139 metadata"
},
{
"type": "application/ld+json"
"title": "JSON-LD metadata"
},
{
"type": "application/ld+json;profile=https://stacspec.org"
"title": "STAC metadata"
},
{
"type": "application/ld+json;profile=https://schema.org"
"title": "JSON-LD metadata"
}
"title": "JSON-LD (schema.org) metadata"},
{
"type": "application/ld+json;profile="http://data.europa.eu/930/",
"title": "JSON-LD (GeoDCAT-AP) metadata"
},
{
"type": "application/rdf+xml",
"title": "RDF/XML metadata"
},
{
"type": "application/rdf+xml;profile=https://schema.org",
"title": "RDF/XML (schema.org) metadata"
},
{
"type": "application/rdf+xml;profile="http://data.europa.eu/930/",
"title": "RDF/XML (GeoDCAT-AP) metadata"
},
{
"type": "text/turtle",
"title": "Turtle metadata"
},
{
"type": "text/turtle;profile="https://schema.org/"
"title": "Turtle (schema.org) metadata"
},
{
"type": "text/turtle;profile="http://data.europa.eu/930/"
"title": "Turtle (GeoDCAT-AP) metadata"
},
{
"type": "text/html",
"title": "HTML"


Example: 3.8
> JWS/CT (Clear Text) signature validation: The signature is included in the original JSON metadata record (payload). See https://www.ietf.org/archive/id/draft-jordan-jws-ct-08.txt.

# jstr is signed JSON metadata
data = json.loads(jstr)

# Fetch the Signature Property String containing the JWS token without a payload part.
sig = data['signature']

Figure A.122

Figure A.123

Figure A.124
Remove the Signature Property String

def data['signature']

Canonicalize the Remaining JSON Object

payload = str(jcs.canonicalize(data))

Validate the JWS String (Annex F of JWS RFC7515)

jwtoken = jws.JWS(payload = None)
jwtoken.deserialized(raw_jws = sig)

try:
jwtoken.verify(key = ss_key.public(), detached_payload = payload)
    print("JWS signature verification completed.")
except jws.InvalidJWSSignature:
    print("JWS signature verification failed.")

Figure A.125

JWS signature verification completed.

Figure A.126

A.10. D112 (CSW) GetCapabilities without DCS


Example: 4.1

> Access the GMU CSW D112 (GetCapabilities) to retrieve supported queryables.

csw_endpoint_items = endpoint_csw + "?SERVICE=CSW&REQUEST=
GetCapabilities&VERSION=2.0.2&sections=OperationsMetadata"

response = requests.get(csw_endpoint_items,
    verify=True,
    headers={})

Extract information about GetRecords from the response.

rt = ElementTree.fromstring(response.text)
opengis.net/ows}Operation[@name='GetRecords']")

try:
xmldata = ElementTree.tostring(r, encoding='unicode', method='xml')
except:
xmldata = 'Not found.'

md("````xml
" + xmldata + "```"")

Figure A.127

```xml
<ns0:Operation xmlns:ns0="http://www.opengis.net/ows" xmlns:ns1="http://www.w3.org/1999/xlink" name="GetRecords"><ns0:DCP><ns0:HTTP><ns0:Get ns1:type="simple" ns1:href="https://cat.csiss.gmu.edu/ows18" /><ns0:Post ns1:type="simple" ns1:href="https://cat.csiss.gmu.edu/ows18" /></ns0:HTTP></ns0:DCP><ns0:Parameter name="CONSTRAINTLANGUAGE"><ns0:Value>CQL_TEXT</ns0:Value><ns0:Value>FILTER</ns0:Value><ns0:Value>brief</ns0:Value><ns0:Value>full</ns0:Value></ns0:Value><ns0:Parameter name="ElementSetName"><ns0:Value>brief</ns0:Value><ns0:Value>full</ns0:Value><ns0:Value>summary</ns0:Value></ns0:Parameter><ns0:Parameter name="outputFormat"><ns0:Value>application/json</ns0:Value><ns0:Value>application/xml</ns0:Value></ns0:Parameter><ns0:Parameter name="FORMAT">
A.11. D112 (CSW) GetRecords without DCS

Example: 4.2
> Access the GMU CSW D112 (GetRecords).

csw_endpoint_items = endpoint_csw + "?SERVICE=CSW&REQUEST=GetRecords&VERSION=2.0.2&ElementSetName=full&typeNames=gmd:MD_Metadata&resultType=results&outputSchema=http://www.isotc211.org/2005/gmd&maxRecords=1"

response = requests.get(csw_endpoint_items,
    verify=True,
    headers={})
# headers={'Accept': 'application/xml'})

xmlstr = minidom.parseString(response.text).toprettyxml(indent='  ',newl='')

md("```xml

+ xmlstr +

```

Figure A.129

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<csw:GetRecordsResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dct="http://purl.org/dc/terms/"
    xmlns:gmd="http://www.isotc211.org/2005/gmd"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ows="http://www.opengis.net/ows"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gco="http://www.isotc211.org/2005/gco"
    version="2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schemas.opengis.net/csw/2.0.2/CSW-discovery.xsd">
    <csw:SearchStatus
        timestamp="2022-10-21T15:19:25Z"/>
    <csw:SearchResults
        numberOfRecordsMatched="8"
        numberOfRecordsReturned="1"
        nextRecord="2"
        recordSchema="http://www.isotc211.org/2005/gmd" elementSet="full">
        <gmd:MD_Metadata
            xsi:schemaLocation="http://www.isotc211.org/2005/gmd http://schemas.opengis.net/csw/2.0.2/profiles/apiso/1.0.0/apiso.xsd"
            xsi:schemaLocation="http://www.isotc211.org/2005/gmd http://schemas.opengis.net/csw/2.0.2/profiles/apiso/1.0.0/apiso.xsd">
            <gmd:fileIdentifier>
                <gco:CharacterString>urn:uuid:b1e7ef9a-0146-11ed-b12e-d19c2be36fcc</gco:CharacterString>
            </gmd:fileIdentifier>
            <gmd:language>
                <gco:CharacterString>en-US</gco:CharacterString>
            </gmd:language>
            <gmd:hierarchyLevel>
                <gmd:MD_ScopeCode codeSpace="ISOTC211/19115" codeList="http://www.isotc211.org/2005/resources/Codelist/gmxCodelists.xml#MD_ScopeCode" codeListValue="vector digital data">vector digital data</gmd:MD_ScopeCode>
            </gmd:hierarchyLevel>
            <gmd:contact>
                <gmd:CI_ResponsibleParty>
                    <gmd:organisationName>
                        <gco:CharacterString>Earth Data Analysis Center</gco:CharacterString>
                    </gmd:organisationName>
                    <gmd:CI_ResponsibleParty>
                        <gmd:organisationName>
                            <gco:CharacterString>Earth Data Analysis Center</gco:CharacterString>
                        </gmd:organisationName>
                        <gmd: CI_ResponsibleParty>
                            <gmd:organisationName>
                                <gco:CharacterString>Earth Data Analysis Center</gco:CharacterString>
                            </gmd:organisationName>
                            <gmd: DATE/>
                            <gmd:dateStamp>
                                <gmd:metadataStandardName>
                                    <gco:CharacterString>ISO19115</gco:CharacterString>
                                </gmd:metadataStandardName>
                                <gmd:metadataStandardVersion>
                                    <gco:CharacterString>2003/Cor. 1:2006</gco:CharacterString>
                                </gmd:metadataStandardVersion>
                                <gmd:identificationInfo>
                                    <gmd:MD_DataIdentification
                                        id="urn:uuid:b1e7ef9a-0146-11ed-b12e-d19c2be36fcc">
                                        <gmd:citation>
                                            <gmd:CI_Citation>
                                                <gmd:title>
                                                    <gco:CharacterString>Current Core Based Statistical Areas for San Miguel County, New Mexico, 2006se TIGER</gco:CharacterString>
                                                </gmd:title>
                                                <gmd:CI_Date>
                                                    <gmd:date>
                                                        <gmd:dateType>
                                                            <gmd:CI_DateTypeCode codeSpace="ISOTC211/19115" codeList="http://www.isotc211.org/2005/resources/Codelist/gmxCodelists.xml#CI_DateTypeCode" codeListValue="publication">publication</gmd:CI_DateTypeCode>
                                                        </gmd:dateType>
                                                        <gmd:CI_Citation>
                                                            <gmd:title>
                                                                <gco:CharacterString>The 2006 Second Edition TIGER/Line files are an extract of selected geographic and cartographic information from the Census TIGER database. The geographic coverage for a single TIGER/Line file is a county or statistical equivalent entity, with the coverage area based on the latest available governmental unit boundaries. The Census TIGER database represents a seamless national file with no overlaps or gaps between parts. However, each county-based TIGER/Line file is designed to stand alone as an independent data set or the files can be combined to cover the whole</gco:CharacterString>
                                                            </gmd:title>
                                                        </gmd:CI_Citation>
                                                    </gmd:date>
                                                </gmd:CI_Date>
                                            </gmd: CI_Citation>
                                        </gmd:MD_DataIdentification>
                                    </gmd:identificationInfo>
                                </gmd:MD_DataIdentification>
                            </gmd:CI_ResponsibleParty>
                        </gmd:CI_ResponsibleParty>
                    </gmd:CI_ResponsibleParty>
                </gmd:contact>
            </gmd:MD_Metadata>
        </csw:SearchResults>
    </csw:GetRecordsResponse>
This shapefile represents the current Core Based Statistical Areas in the 2006 TIGER Second Edition dataset for San Miguel County, NM. The 2006 Second Edition TIGER/Line files consist of line segments representing physical features and governmental and statistical boundaries.
A.12. D112 (CSW) GetRecordById without DCS

Example: 4.3
> Access the GMU CSW D112 (GetRecordById).

```
item_id = "urn:uuid:b1e7ef9a-0146-11ed-b12e-d19c2be36fcc"
csw_endpoint_item = endpoint_csw + '?SERVICE=CSW&REQUEST=GetRecordById&VERSION=2.0.2&FORMAT=application/xml'
response = requests.get(csw_endpoint_item, verify=True, headers={})
```

```python
# Extract ISO19139 metadata record from the response.
rt = ElementTree.fromstring(response.text)
```

```python
try:
xmxtxt = ElementTree.tostring(r, encoding='unicode', method='xml')
except:
xmxtxt = 'Not found.'
```

```
md("```xml
... + xmxtxt + "```")
```

Figure A.131

```
```

```
<ns0:CI_ResponsibleParty>
    <ns0:organisationName>Earth Data Analysis Center</ns0:organisationName>
    ...
</ns0:CI_ResponsibleParty>
<ns0:contact>
    ...
</ns0:contact>
```

The 2006 Second Edition TIGER/Line files are an extract of selected geographic
and cartographic information from the Census TIGER database. The geographic coverage for a single TIGER/Line file is a county or statistical equivalent entity, with the coverage area based on the latest available governmental unit boundaries. The Census TIGER database represents a seamless national file with no overlaps or gaps between parts. However, each county-based TIGER/Line file is designed to stand alone as an independent data set or the files can be combined to cover the whole Nation. The 2006 Second Edition TIGER/Line files consist of line segments representing physical features and governmental and statistical boundaries.

This shapefile represents the current Core Based Statistical Areas in the 2006 TIGER Second Edition dataset for San Miguel County, NM.
### A.13. Subscriptions to OGC API-Records

See information available at [https://ogc.demo.secure-dimensions.de/sms/api](https://ogc.demo.secure-dimensions.de/sms/api).


```python
srv_kid = "srvkid"
cli_kid = "clikid"
ss_key = jwk.JWK(generate='RSA', kid=srv_kid, use="sig")
se_key = jwk.JWK(generate='RSA', kid=srv_kid, use="enc")

# client key pair for encryption (public+private)
ce_key = jwk.JWK(generate='RSA', kid=cli_kid, use="enc")
```

**Figure A.133**

```
"d": "_rw1-M-daU3gW00Iuhltk3uFpvo0lttiSbeIa_A1Gn75mc48w3kaTXH0RHpC6whAz88N0fKxbJwDbmsUs6Pjzj-Qg5odBuUly172S8a5OpKMF0eivcYZXy2s-L4_attPMJSyGMrzTzhp5jtmzVTSSESuD-KYkJPfTbbA7UVBt43usuNTUhxDVcQY-emGTAYsicOeZGWFfWYO24LjX1RKp95IztCa0MrwM2MPE8rA1muVkkk3MnhhddOLRTlw0TexpAwnG
bTMHlwBkRoBjo47BF0PwnX_MtepHzvQA36roXkimS_80C.PR4DqPa0up3Db0s6B10d0xwBRkcYXOC
zY4Gp-xochHHPAOnrrj96eaEsWHMBo9zArFxKIMLpzDh24D0NpxV2h6hjQAP-
dheSYFRwcQ0F5uj1NkFVvTf19Uty1PbKTKNm9exMSwAEXe14bJlfFqS89k6NGsFTEuhVnavf7e
iTu","e":"AQAB","kid":"clikid","kty":"RSA","n":"2qBsAlkltTmv-
oMoLVsasay9B-CAo_3ie5fi8Mg9aph72w9uR5gEcfz6A-rsPLMh8BuNwuUIoK073NRc-
gtXBA9UXUSvNn86djiHTEyKAZmYbmb3_qZGSd10KlgqDnx-qG0i2r5JSN3AQYIaLVili_q5aIVDowKQjY7NZR1C7Z2L7jqQfSeIOISt_K57Gt9wo7GIHcu00nzt3hldoPwQiVpn04Bz14PXaDFB4p8d3BcrLwKz3ESy6sW6CiZmIQw","p":-uTjBer1iYAs1L11ussn18763ki0f10K5Lcj_r2nFZ1-
qwd52fQzSoIkSKJ062ZmIYBuVLzF7P4mkj_NmWbakxmLAdR2jIsQG8BU1IdykhzhrlL7yYmBQ
zfB_KoZ6XH2aAkUCriKb61wsHohG3Q_tJhMtqiSIG4jcuW1cqvTv6x4uYsdPfycwI-
kslcTcfQ8ChycZ2dTMrQR0ycgb-
Xct5P1JFAV0C","qi":"H7sbm1Vp7BLX5gihmu5sA3H2nf-
NyG6M01ypYBcR5RpyaouoEQVEQQV7k13Gm1Wt3jYFvQ79hNHzY6UYVOiLV-
U4OFRxZw2AkQMjQsQrdddjlgQ8hRJ_S_GogFnxAgd80m2cdX1LZGoECFsEy1QLjUHs_fw
```

**Figure A.134**

```python
# Prepare subscription object including public key.  Put here your actual
e-mail address to receive notifications.

myobj = {
    "name": "pySubscription",
    "delivery": "mailto:john.doe@company.com",
    "schedule": "* * */1 * *",
    # "resources-uri": endpoint_items + '?f=xml',
    "resources-uri": endpoint_items,
    "expires": 1759104237,
    "sec-opts": {
        "x-ogc-callback-key": "string",
        "public-key": ce_key.export()
    }
}
```
myobj

```
Figure A.135

```

```
Figure A.136

```

```
Figure A.137

```

```
Figure A.138

```

### A.13.1. Create subscription

Example: 5.1

> Create subscription.

```python
response = requests.post(endpoint_subscriptions, json = myobj,
    verify=bool(verify_ssl),
    headers={"Accept": 'application/json', "Authorization": 'Bearer ' +
access_token})
```

```
response

```

```
<Response [201]>

```

### A.13.2. Get list of subscriptions

Example: 5.2
Get list of subscriptions.

```python
response = requests.get(endpoint_subscriptions,
                         verify=bool(verify_ssl),
                         headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})

response.text
```

Figure A.139

```json
[{
  "id":212,
  "name": "pySubscription",
  "delivery": "mailto:john.doe@company.com",
  "schedule": "* *1 * * *",
  "resources-uri": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items",
  "expires": 1759104237,
  "sec-opts": {
    "x-ogc-callback-key": "string",
    "public-key": {
      "d": "_rw1-MdaUJgW80IuhltH3uFpvOlvttsiBea_A1Gn75m4c8w3kaTHX0RHpc6wA288NFkxbJwDbmsUs6PJz-jq-Gq5odBUyij7258aSoPkmFoiewyczXWS-L4_attPMJSdGmXtZhppTjmzVtSsesuD-kykJkpTbBvA7US64v3usNTUhxDvcQy-emGTAYsicOeZGWFGWYo24LjX1RKP95IztCa0Mw2MPEPE8r1Amuvffk3MnhntdOLRtLw0TexpAwNg",
      "p": "C2xPnUS6Jkue69unpL4bfrXfQ7aDnDwqdwaid1S10aPiaxL10W4dO6vryq0t134w3o760Qj",
      "kid": "cikid",
      "kty": "RSA",
      "n": "tt88JhkKsd7r7zY7mZGCxMVIQs2uWudufE5K3BFq-6rvvyf1eHoroKpcqlLVBmSD3C9pyv8ML0n7r5ZWPZpeMJt

nSeIu197A-3ex29NjIbnXW7l2xIeu-_a5IJCZDjhrXU00y8blzz2BwYy20SLC79W0FB-gklANm_zzvBMWanT6g3Ixmzf_YldZyVl137UVzR0dDb5ldK02s9fdQfbyGmppp25BCx5w76JueU7
gTYxb1NIC55Mz-ORYXmSZEO-rtOM53bck08UrEz4Qsopw",
      "p": "xprCRD94cvlh5wigJWQVXs3z-KZG1l0feE5h4aDebeigRgtPmu12RPMaB4uiuJ3c1i1MnjLgfo1impeM_y_uMe8jGnhnkrxJfrF800y_kRRT6NWCJHtFDxalK9uSoUuWnxmL3PceQCq5jHfhKpg4v8t0r",
      "q": "617l92Ux744sx9WkfNKe1UNB4p07qJxChwV1u-sfE_J5_Vtt8ctHd07oSoQr9mLe_klpEANxYQXNxdn59vRHZMB1JGnKua5CQFXBxPL39gbKkK

\n"},
```

```python
'verify=verify_ssl",
headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})
```

response = requests.get(endpoint_subscriptions,
                         verify=True,
                         headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})

response.text
```

Figure A.139

```json
[{
  "id":212,
  "name": "pySubscription",
  "delivery": "mailto:john.doe@company.com",
  "schedule": "* *1 * * *",
  "resources-uri": "https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items",
  "expires": 1759104237,
  "sec-opts": {
    "x-ogc-callback-key": "string",
    "public-key": {
      "d": "_rw1-MdaUJgW80IuhltH3uFpvOlvttsiBea_A1Gn75m4c8w3kaTHX0RHpc6wA288NFkxbJwDbmsUs6PJz-jq-Gq5odBUyij7258aSoPkmFoiewyczXWS-L4_attPMJSdGmXtZhppTjmzVtSsesuD-kykJkpTbBvA7US64v3usNTUhxDvcQy-emGTAYsicOeZGWFGWYo24LjX1RKP95IztCa0Mw2MPEPE8r1Amuvffk3MnhntdOLRtLw0TexpAwNg",
      "p": "C2xPnUS6Jkue69unpL4bfrXfQ7aDnDwqdwaid1S10aPiaxL10W4dO6vryq0t134w3o760Qj",
      "kid": "cikid",
      "kty": "RSA",
      "n": "tt88JhkKsd7r7zY7mZGCxMVIQs2uWudufE5K3BFq-6rvvyf1eHoroKpcqlLVBmSD3C9pyv8ML0n7r5ZWPZpeMJt

nSeIu197A-3ex29NjIbnXW7l2xIeu-_a5IJCZDjhrXU00y8blzz2BwYy20SLC79W0FB-gklANm_zzvBMWanT6g3Ixmzf_YldZyVl137UVzR0dDb5ldK02s9fdQfbyGmppp25BCx5w76JueU7
gTYxb1NIC55Mz-ORYXmSZEO-rtOM53bck08UrEz4Qsopw",
      "p": "xprCRD94cvlh5wigJWQVXs3z-KZG1l0feE5h4aDebeigRgtPmu12RPMaB4uiuJ3c1i1MnjLgfo1impeM_y_uMe8jGnhnkrxJfrF800y_kRRT6NWCJHtFDxalK9uSoUuWnxmL3PceQCq5jHfhKpg4v8t0r",
      "q": "617l92Ux744sx9WkfNKe1UNB4p07qJxChwV1u-sfE_J5_Vtt8ctHd07oSoQr9mLe_klpEANxYQXNxdn59vRHZMB1JGnKua5CQFXBxPL39gbKkK

\n"},
```

```python
'verify=verify_ssl",
headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})
```

response = requests.get(endpoint_subscriptions,
                         verify=True,
                         headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})

response.text
```
"kty":"RSA","n":"2qBsAlktTmv-oMoLVsay9B-CAo_3ie5fi8Mg9Aph72w9uR5gEcfz6A-rsPLMH8buNwuUioKo73NRC-glXBA9UXUsVnN86djIHTEyKAZm9ybm3_q2GSDl0KlgqDnx-qGOi2r5JSN3AQYLAvLi1_qSaIVDowKQJY7N2R1C7Zz7jqQfszEoIRSt_K57Gt9wo7GIHuOnz3hldoPwQiVpBno4Bz14PxaDFB4p8d3BcrLwKz3ESy6sW6CiZm1Qw",
"p":"-uTjbErAYsiL11usnn1B763ki0fl0K5Lcjr_r2nFZ1-qwd52fZgSoIKsKJ06zPmIkYSuVDLzfzTP4mKj_NmWBakxmLAdR2jIsGQ8BU1IdkzyzhgrL7vYmbQ",
"q":"3xNtcpbvk3RUDBgjoxhB1HzKvNSDDTYGf0C2HNil9snjvF4mp2CfTe7wRd7i7N5_R-ziel_KoZ6XHkaAKuCRiKb61wsIohG3QtJhMrtqiSIG4jcuW1cqVt6X4uYsDpFyeEcxiI-ksltcf8ChycZDTamXROcygb-Xct5P1jFAVOC",
"qi":":H7sbn1Vp7BlX5ghmu5s4A3H2nf-NyG6M01pYBcr5RpyauwEQQEVQK71JGm1Wt3jYFVQ79wNhYz6UYVoilLV-U4oFRXz2wAKMQjQvrQrQrdujlgQ8hRj_S_6ogQnFxAGd8onm2cdX1LZGoEPCfsEy1QLjUhs-fw",
"use":"enc"}
'

Figure A.140

# extract first subscription id

data = json.loads(response.text)
id = data[0]['id']

Figure A.141

212

Figure A.142

A.13.3. Get subscription status

Example: 5.3

> Get subscription status.

```python
response = requests.get(endpoint_subscriptions + ""+ str(id),
                       verify=bool(verify_ssl),
                       headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})
```  

response.text

Figure A.143

'"name":"pySubscription","delivery":"mailto:john.doe@company.com","schedule":
* */1 * *","resources-uri":"https://ogc.demo.secure-dimensions.de/pycsotentagets/main
"expires":1759104237,"sec
"x-ogc-callback-key":"string","public-key":{"d":"string","public-key":{"d":"string"}}
"GTSwJzYoVf79k7B6Gw3iiTy_VnFHL2PFiVcTbRSs_TnEN0_AgoUPZQ0U2gKM1L2vTiNqbAg1-
haQIKpMUnLy432IC3TLZ-f1tW5-0YQR3zk2qdsQ5rIqFpsBjBjCJd4F9ifxTQeeYhaAkdh-
SBmo2k1kwkAw_g4PH3fQdfM57R4iuy32TF6u-uLT5p54qRuQ-
TWZOB_vzPQZ7jVc6skNvW-
XNmNtg2hy9fS4eOk1iO7f47QZyIKBSywKGAresxKfVnBuEppQEvdZKbS"
```

Figure A.144
A.13.4. Start this subscription

A subscription is started by updating its state to `start`. State transitions needed are created → approved → ready → started. State becomes approved when the code received by email at creation time is used in the PATCH. Status becomes `ready` when the resource-uri is available as well.

Example: 5.4

> Approve subscription (PATCH).

```python
myobj = {
    # use here the code you received by email for this subscription id.
    "code": "578Z8C"
}
```

response = requests.patch(endpoint_subscriptions + "/" + str(id), json = myobj, verify=bool(verify_ssl), headers={"Accept": 'application/json', 'Authorization': 'Bearer ' + access_token})

Figure A.145

<Response [400]>

Figure A.146

Example: 5.5

> Start subscription (PATCH).

Use PATCH to change the subscription from state `ready` → `started`.

```python
myobj = {
    "state": "start"
}
```
response = requests.patch(endpoint_subscriptions + "" + str(id), json = myobj, verify=bool(verify_ssl), headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})

response

Figure A.147

<Response [400]>

Figure A.148

Check that state is updated to `start`.

response = requests.get(endpoint_subscriptions + "" + str(id), verify=bool(verify_ssl), headers={'Accept': 'application/json', 'Authorization': 'Bearer ' + access_token})

response.text

Figure A.149

'{"name":"pySubscription","delivery":"mailto:john.doe@company.com","schedule":null,"resources-uri":"https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items","expires":1759104237,"sec-opts":{"x-ogc-callback-key":"string","public-key":null},"state":"created","created":1666365527}

Figure A.150
A.13.5. Delete the subscription

Example: 5.6
> Remove subscription (DELETE).

```python
# id = 170
response = requests.delete(endpoint_subscriptions + "/" + str(id),
   verify=bool(verify_ssl),
   headers={'Authorization': 'Bearer ' + access_token})
```

![Figure A.151](image1.png)

![Figure A.152](image2.png)

A.13.6. Get updated subscription list

```python
response = requests.get(endpoint_subscriptions,
   verify=bool(verify_ssl),
   headers={'Accept': 'application/dcs+geo', 'Authorization': 'Bearer ' + access_token})
```

```python
response.text
```

```
'[
{"id":213,"name":"pySubscription","delivery":"mailto:john.doe@company.com","schedule":"* */1 * *","resources-uri":"https://ogc.demo.secure-dimensions.de/pycsw/collections/metadata:main/items","expires":1759104237,"sec-opts":{"x-ogc-callback-key":"string","public-key":"\"d\":\"_rw1-M-daUjgW80IuhtH3uFpv0lttiSbe1a_A1Gn75mc48w3kaTXH0RHpC6whA288N0fKxbjWDbmUs6PJzj-Gq5odBUyi725S8a0QKMF0eicwYZxWs-L4_attPMJ5DgMxTzHtpTjmxTzSStSEsUdDKYkJTPbvbA7UVBIt43usuNTUhxDvCqY-emGTAYsic0eZGWFWYo24LjX1RKP951ztCa0Mw2MPEPE8rA1muvevkk3Mnhntd0LRTlw0TexpAwNG
","dp":":"\"P60BtUnExG1zxR6obqGnP5z2yCGtp0LXjrg5tFJa6H0PV_wALbPSfEf67dj81vQTmZnPvmoTbTMHlwBkR0BjO47BF0PHwN_X_MtepHzvYA36roXkimS_80c_P4DqPpao35dbs6Bid0xwBRRcsYXC
","dq":":"\"TQ466ggzsiLTQK3VTGk1J-y4Gp-xochHPALnrrj96aeHUWMb09zArFxKIMLpaDHs24DN0PxyVC2h6hJQAP-dheSYFRwcQOF5uj1NkfFvyTf19hTzPp8KTTNm9eXMSwAEVe14bJlFqFs89k6NgFsTEuhVnavf7eitU","e":"\"AQAB\"","kid":":\"clikid\",\"kty\":":\"RSA\",\"n":":\"2qBslkltTmv-oMoLVsawy9B-CAo_3ie5f18Mg9Ap7h2w9uR5gEcfz6A-rsPLM8buNUwU0ko73Nrc-glXB98UXVSn86dJ1ihTEyKAZm9yym3_qZGSDl0KlqgDnx-qG0i2r5JNS3AQQY1AvL1I_qSaIvD0wKQJY7NZR1C722l7jQFSAeEoIRSt_K57Gt9wo7GIHcuOnzu3hldoPwQiVpBoa4Bz14PxaDFB4p8d3BcrLwKz3ESy6sW6CiZmIqw\"","p":":\"-uTJeriyA711L1sn1n1B763k10fl0K5Lcjr2nFZ1-qwd52ZqSoIKsKJ06zPmIKYSuVDLzfzTP4mkj_NmWBakxMAdR2jIsGQ8BU1IdkzyhrL7vYmbQ
```


Figure A.154
ANNEX B (INFORMATIVE) REVISION HISTORY
# ANNEX B

(INFORMATIVE)

REVISION HISTORY

<table>
<thead>
<tr>
<th>DATE</th>
<th>RELEASE</th>
<th>AUTHOR</th>
<th>PRIMARY CLAUSES MODIFIED</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-06-02</td>
<td>0.1</td>
<td>Y. Coene</td>
<td>all</td>
<td>Initial draft version</td>
</tr>
<tr>
<td>2022-10-21</td>
<td>0.8</td>
<td>Y. Coene</td>
<td>all</td>
<td>Version addressing review comments from Carl Reed and agentschap Digitaal Vlaanderen.</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY
BIBLIOGRAPHY


