Draft Outline for OGC API - Catalogue

INTRODUCTION

- current title: "OGC API - Catalogue - Part 1: Core"
- a catalogue is a curated collection of metadata records about resources
- a resource can be anything that can be discovered
  - e.g. features, coverages, assets, services, widgets, etc.
- a catalogue may implement any underlying data model that it wants (e.g. ebRIM, ISO19115/ISO19119, etc.) however, all internal data models must be mapped to the record schema defined in this standard
  - of course, a catalogue implementation may choose to directly implement the record schema defined in this standard as its information model
  - this record schemas shall be referred to as the "standard record" in this document
- the standard record is based on DCAT and looks similar to GeoJSON but is not GeoJSON (since GeoJSON is specifically for geographic features)
  - there are also some hints of STAC thrown in for good measure! ;)
- besides the properties defined for the standard record, a server may also expose additional properties that may be used to query the catalogue
  - e.g. specialized catalogues such as imagery catalogues may expose additional query properties such as cloud_cover or albedo
- the catalogue API is based on OGC API suite of APIs (i.e. OGC API - Features)

CONFORMANCE CLASSES

- CAT 4.0 conformance classes are:
  - core
    - read-only catalogue with limited query capability
  - full text search query parameters
    - ability to perform full text search
  - OpenSearch query parameters
    - ability to search the catalogue using extended OpenSearch Geo parameters
  - complex query parameters
    - support for complex query predicates use HTTP query parameters
  - complex query resource
    - ability to search the catalogue using complex predicates
  - classification query parameters
    - ability to perform a taxonomy search on the catalogue
  - transactions

        - ability to add, modify and remove records from the catalogue
    - json-record
        - JSON representation of a standard catalogue record
    - xml-record
        - XML representation of a standard catalogue record
    - html-record
        - HTML representation of a standard catalogue record

CONTENT NEGOTIATION

- done using the Accept and Content-Type HTTP headers
- servers can also mint their own URLs that include a format parameter (e.g. "OGC API - Features" servers use "f" for format)
    - servers, would of course, include any such parameter in their OpenAPI document

LANGUAGE NEGOTIATION

- done using the Accept-Language and Content-Language HTTP headers
- server can also mint their own URLs that include a language parameter
    - (e.g. …&lang=en…)
- languages shall be specified as per RFC 1766 (which is based on ISO 639)
    - e.g. en for English, el for Greek

HTTP 1.1

- servers SHALL conform to HTTP 1.1
- if the server support HTTPS, the server SHALL conform to HTTP over TLS

EXCEPTIONS

- clients should be prepared to handle the following HTTP status codes
- 200, 304, 400, 401, 403, 404, 405, 405 and 500
- server SHOULD include an exception body with more information

Example exception report

```
{
   "version": "2.0.2",
   "exceptions": [
      {
         "code": "999",
         "locator": "insert stmt 01",
         "text": "parse error: missing closing tag"
      }
   ]
}
```

HTTP QUERY PARAMETERS

- if a URL includes unknown or invalid query parameters the server SHALL throw an exception
- all parameters (specified in the standard or vendor specific) should be defined in the server's OpenAPI document

OTHER CONSIDERATIONS

- ETAGS for web caching
- CORS

ENCODINGS

- no mandatory encodings are defined by this standard
- this standard, however, defines 3 output format conformance classes
  - JSON, XML and HTML

WEB LINKING

- "links" sections are included all over the place in response messages defined in this standard to allow web linking
- "links" in payloads SHOULD also be included in the headers using the Link header
  - this is only a recommendation because in some cases the number of links in the payload may be large and thus not feasibly included in the headers or the links may not be known at the time the headers are being written

RESOURCE PATHS

Table 1. API resource paths

Path	Description
/	Landing page
/api	API definition
/conformance	Conformance declaration
/collections	List of available catalogues
/collections/{catalogueId}	Metadata about a catalogue
/collections/{catalogueId}/items	Access path to catalogue records
/collections/{catalogueId}/items/{recordId}	Access to a specific record
/collections/{catalogueId}/queryables	List of catalogue's queryables (i.e. record properties that can be query predicate)

METHODS (by path)

Table 2. HTTP methods per resource path

Path	Method	Description	Conformance class
all	HEAD	returns only the HTTP headers for the specified resource (i.e. GET without the body)	
	OPTIONS	gets methods and representations for the specified resource	
/	GET	get the server's landing page	core
	PUT	undefined	
	POST	undefined	
	PATCH	undefined	
	DELETE	undefined	
/api	GET	get the server's API definition document (i.e. OpenAPI doc)	
	PUT	undefined	
	POST	undefined	
	PATCH	undefined	
	DELETE	undefined	
/conformance	GET	get the server's conformance declaration document	
	PUT	undefined	
	POST	undefined	
	PATCH	undefined	
	DELETE	undefined	

Path	Method	Description	Conformance class
/collections	GET	get the list of metadata record collections (i.e. catalogues)	
	PUT	undefined	
	POST	undefined (ext: create a new collection?)	
	PATCH	undefined	
	DELETE	undefined	
/collections/{catalogueId}	GET	get the record describing the specified catalogue	core
	PUT	undefined (ext: update metadata about the catalogue)	
	POST	undefined	
	PATCH	undefined (ext: update metadata about the catalogue)	
	DELETE	undefined (ext: delete the catalogue)	
/collections/{catalogueId}/items	GET	retrieve a subset of records from the catalogue	core
	PUT	undefined	
	POST	add a new metadata record to the catalogue	tx
	PATCH	undefined	
	DELETE	undefined	
/collections/{catalogueId}/items/{recordId}	GET	retrieve a specific catalogue record	core
	PUT	replace the specified catalogue record with the one in the request body	tx
	POST	undefined	
	PATCH	updates a portion of a catalogue record	tx
	DELETE	remove the specified record from the catalogue	tx
/collections/{catalogueId}/queryables	GET	get the list of queryables for the specified catalogue	
	PUT	undefined	
	POST	undefined	
	PATCH	undefined	
	DELETE	undefined	

/ path

- as per "OGC API - Features"

/api path

- as per "OGC API - Features"

/conformance path

- as per "OGC API - Features" (with CAT 4.0 conformance classes of course)

/collections

- a catalogue is a "collection" of metadata records
- a catalogue end point may offer a SINGLE collection of metadata records or it may offer multiple collections of metadata records
    - e.g. a feature catalogue and an imagery catalogue
- a CAT 4.0 catalogue shall offer at least ONE collection of metadata records
- a successful GET operation on the /collection path shall return a response with a 200 HTTP status code
- the content of the response shall be an array of objects each describing a catalogue that is available at this end point
- the content of the response SHALL a self link (i.e. rel="self") and zero or more links (rel="alternate") pointing to the alternate representations of the response that the server support (e.g. HTML, XML, etc.)

Example /collections response

```
GET /collections
{
   "links": [
      {
         "href": ".../collection?f=json",
         "rel": "self",
         "type": "text/json"
      },
      {
         "href": ".../collection?f=xml",
         "rel": "alternate",
         "type": "text/xml"
      },
      {
         "href": ".../collection?f=html",
         "rel": "alternate",
         "type": "text/html"
      }
   ],
   "collections": [
      {
         "id": "ogc_catalogue",
         "type": "catalogue",
         "title": "OGC Catalogue",
         "description": "A catalogues of OGC resources and services.",
         "language": "en",
         "issued": "2019-01-01",
         "modified": "2019-05-21",
         "properties": {
            "publisher": "CubeWerx Inc.",
            "license": "Some legal gibberish about terms of use...",
            "rights": "More legal gibberish about usage rights...",
         },
         "links": [
            {
               "href":
"http://demo.cubewerx.com/cubewerx/cubeserv/default/csw/4.0/collections/ogc_catalogue/
search",
               "rel": "search",
               "title": "Complex Search Endpoint"
            }
         ]
      }
   ], ...
}
```

/collections/{catalogueId}

- metadata about a single collection of metadata records (i.e. catalogue)
- the available {catalogueId} values are the set of "identifier" values in the /collections response (i.e. $.collections[*].id)
- a successful GET operation on the /collection/{catalogueId} path shall return a response with a 200 HTTP status code
- the content of the response shall be a document containing metadata about the catalogue
- the response here should be the same as the metadata about the catalogue presented in the /collections response

Example /collections/{catalogueId} response

```
    GET /collections/ogc_catalogue
    {
       "id": "ogc_catalogue",
       "type": "catalogue",
       "title": "OGC Catalogue",
       "description": "A catalogues of OGC resources and services.",
       "language": "en",
       "issued": "2019-01-01",
       "modified": "2019-05-21",

       "geometry": {
          "type": "Polygon",
          "coordinates": [ ... ]
       },
       "properties": {
          "publisher": "CubeWerx Inc.",
          "license": "Some legal giberish about terms of use...",
          "rights": "More legal giberish about usage rights...",
       },
       "extents": [
          {
             "spatial": {
                "bbox": [44.7972,-140.2037,61.9909,-5.4890],
                "crs": "http://www.opengis.net/def/crs/EPSG/0/4326"
             },
             "temporal": {
                "interval": ["2019-01-01","2019-05-21"],
             }
          }
       ],
       "links": [
          {
             "href":
"http://demo.cubewerx.com/cubewerx/cubeserv/default/csw/4.0/collections/ogc_catalogue/
search",
             "rel": "search",
             "title": "Complex Search Endpoint"
          }
       ]
    }
```

/collections/{catalogueId}/items/{recordId}

- access path for a single record in a catalogue
- the available {catalogueId} values are the set of "identifier" values in the /collections response (i.e. $.collections[*].id)

- a successful GET on the /collection/{catalogueId}/items/{recordId} path shall return a response with a 200 HTTP status code
- the content of the response shall be a catalogue record encoded according to the content negotiation performed between the client and the server

Example record

```
      GET /collections/ogc_catalogue/items/36486763-db57-43b1-9af7-b7ecc3c318f2
      {
         "id": "36486763-db57-43b1-9af7-b7ecc3c318f2",
         "title": "The Resources Title",
         "description": "Some human readable description of the resource.",
         "language": "en",
         "type": "some_resource_type",
         "geometry": {
            "type": "Polygon",
            "coordinates": [
               [
                  [-10.0, -10.0],[-5.0, 0.0],[0.0, 0.0],
                  [10.0, 10.0],[-6.0, -7.0],[-10.0, -10.0]
               ]
            ]
         },
         "extents": [
            {
               "spatial": {
                  "bbox": [-10.0,-10.0,10.0,10.0],
                  "crs": "http://www.opengis.net/def/crs/EPSG/0/4326"
               },
               "temporal": {
                  "interval": ["2019-05-21T07:05:35","2019-05-21T07:07:08"],
                  "trs": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
               }
            }
         ],
         "links": [
            {
               "href":
"http://demo.cubewerx.com/cubewerx/cubeserv/default/csw/4.0/ogc_catalogue/RID574",
               "rel": "related",
               "title": "Some related record in this catalogue.

            }
         ]
      }
```

/collections/{catalogueId}/items

- the access path for the collection of records
- the following parameters may be specified on the /collections/{catalogueId}/items path: limit, bbox, datetime and prop=value for property filtering
    - see "OGC API - Features" for details
- a successful GET operation on the /collection/{catalogueId}/items path shall return a response with a 200 HTTP status code
- the content of the response shall be zero or more catalogue record instances

Example query

```
GET /collections/ogc_catalogue/items?limit=17
{
   "numberMatched": 100,
   "numberReturned": 17,
   "timeStamp": "2019-05-21T13:28:04",
   "links": [ ... ],
   "records": [
      { ... },
      { ... },
      { ... },
      ...
   ]
}
```

FULL TEXT SEARCH CONFORMANCE CLASS

- a parameter to support text searches

Table 3. Full Text search query parameters

PARAMETER	DESCRIPTION
q	full text search (i.e. contains)

OPENSEARCH QUERY CONFORMANCE CLASS

- this standard defines a set of, optional, additional parameters that allow for richer catalogue queries then those supported by "OGC API - Features"
- these parameters are part of the OpenSearch query conformance class
- if a server supports these parameter it SHALL declare, in its conformance document (obtained via the /conformance path), that it supports the OpenSearch query conformance class

Table 4. Extended OpenSearch query parameters

PARAMETER	DESCRIPTION
geometry	WKT geometry
geometry_crs	geometry CRS
gRelation	one of: "intersects", "equals", "disjoint", "touches", "within", "overlaps", "crosses", "contains" (default: intersects)
lat,lon,radius	proximity search
time	as per WFS 3.0
tRelation	one of: "tEquals", "anyInteracts", "after", "before", "begins", "begunBy", "tContains", "during", "endedBy", "ends", "meets", "metBy", "tOverlaps", "overlappedBy" (default: anyInteracts)

COMPLEX QUERY PARAMETERS CONFORMANCE CLASS

- the following HTTP query parameters are defined to support complex query predicates encoded using some predicate language such as CQL

Table 5. Complex query parameters

filter	query predicate in some language
filter_language	language used in filter parameter

COMPLEX QUERY RESOURCE CONFORMANCE CLASS

- if a server supports complex queries via POST then it SHALL declare, in its conformance document (obtains via the /conformance path), that it supports the complex query conformance class
- if a server indicates in its conformance document that it supports complex queries then it SHALL, in the catalogue's metadata (obtained via the /collections or /collection/{cataglogueId} paths), include a link (rel="search") defining the complex query endpoint
- the specific predicate language(s) supported by the server may be determined by using the OPTIONS method on the search endpoint
- besides the properties of the standard record, additional queryables that may be used in predicates can be obtained via the /collections/{collectionId}/queryables path (see below)
- to execute a complex query, a body containing the text of the predicate is POST'ed to the search endpoint
- a JSON-encoding of OGC filter has been developed (for this interested is reusing the OGC filter processor)
    - of course, you can still use the XML-enocded OGC filter if you set the content type header appropriately

Example complex query (using the JSON encoding for OGC filter)

```
POST /collections/ogc_catalogue/search

{
   "and": {
      "isLike": {
         "escapeChar": "\\",
         "singleChar": "?",
         "wildCard": "*",
         "valueReference": "title",
         "literalValue": "*elevation*"
      },
      "=": {
         "valueReference": "type",
         "literalValue": "service"
      },
      ">=": {
         "valueReference": "modified",
         "literalValue": "2004-03-01"
      },
      "intersects": {
         "valueReference": "bbox",
         "geometry": {
            "type": "Polygon",
            "coordinates": [
                [
                    [100.0, 0.0],
                    [101.0, 0.0],
                    [101.0, 1.0],
                    [100.0, 1.0],
                    [100.0, 0.0]
                ]
            ]
         }
      },
      "during": {
         "valueReference": "modified",
         "interval":["2019-01-01","2019-05-21"]
      }
   }
}
```

CLASSIFICATION QUERY CONFORMANCE CLASS

- this class defined parameters that may be used to execute queries against taxonomies used to classify catalogue records

- the parameter are:
  - classifiedAs (type: anyURI)
    - the value of the classifiedAs property is a URI referencing a node is a taxonomy
  - scope (one of: broad, narrow, exact)
    - a scope of "broad" means, find all catalogues records classified as the specified node in the taxonomy and all child nodes of the specified node
    - a scope of "narrow" means, find all catalogues records classified as the specified node in the taxonomy and all parent nodes of the specified node
    - a scope of "exact" means, find all catalogues records classified exactly as the specified node in the taxonomy
- Example: consider the following taxonomy:

```
                  A
                  |
        +---------+---------+
        B                   C
   +----+----+         +----+----+
   |         |         |         |
   D         E         F         G
 /   \     /   \     /   \     /   \
H     I   J     K   L     M   N     O
```

- GET …&classifiedAs=B&scope=broad … will find records classified as B, D, E, H, I, J and K
- GET …&classifiedAs=B&scope=narrow … will find records classified as B and A
- GET …&classifiedAs=B&scope=exact … will find records classified as B

/collections/{catalogueId}/queryables

- access path for get the set of additional queryables from a catalogue
- the available {catalogueId} values are the set of "identifier" values in the /collections response (i.e. $.collections[*].id)
- a successful GET on the /collection/{catalogueId}/query path shall return a response with a 200 HTTP status code
- the content of the response shall be a catalogue record encoded according to the content negotiation performed between the client and the server as per the HTTP rfc

Example queryables request

```
GET /collections/ogc_catalogue/queryables
{
   "links": [ ... ],
   "queryables": [
      {
        "identifier": "platform",
        "title": "Platform",
        "description": "Name of acquisition platform.",
        "datatype": {
           "name": "string",
           "reference": "https://www.w3.org/TR/xmlschema11-2/#string"
        }
      },
      {
        "identifier": "sun_azimuth",
        "title": "Sum Azimuth",
        "description": "The direction of a celestial object from the observer,
expressed as the angular distance from the north or south point of the horizon to the
point at which a vertical circle passing through the object intersects the horizon.",
        "datatype": {
           "name": "double",
           "reference": "https://www.w3.org/TR/xmlschema11-2/#double"
        }
      },...
   ]
}
```

CROSS CATALOGUE QUERIES

- there might be a need to execute queries across all the catalogues offered by an endpoint
- the following paths are defined for this purpose:
  - /items
    - path to all records in all catalogues offered by this endpoint
    - behaves like the /collections/{catalogueId}/items path
  - /items/{recordId}
    - path to a records in all catalogues offered by this endpoint
    - behaves like the /collections/{catalougeId}/items/{recordId} path
  - /queryables
    - returns the set of queryables that may be used across all catalogues
    - behaved like the /collections/{catalogueId}/queryables path

STATIC CATALOGUE

- T.B.D. need to "steal" this concept from STAC