API OF THE ILDG CATALOGS

11/07/2025 | Giovanni Pederiva |

Who is this talk for?

- Anyone who is interested in finding out the inner workings of the ILDG systems
- People who might be interested in developing user tools and clients

Why Standardized APIs?

- The APIs are an essential part of the ILDG, specified by the MWWG
- They ensure the interoperability of the different regional grids
- Different regional grids can have independent implementations of the services
- Regional specific extensions are allowed on top of the minimum requirements set by the ILDG

This is still WIP: Proposed API Specification for ILDG 2.0

A reference implementation, still in development and not fully compliant, has been developed by PUNCH4NFDI

Examples

Client tools:

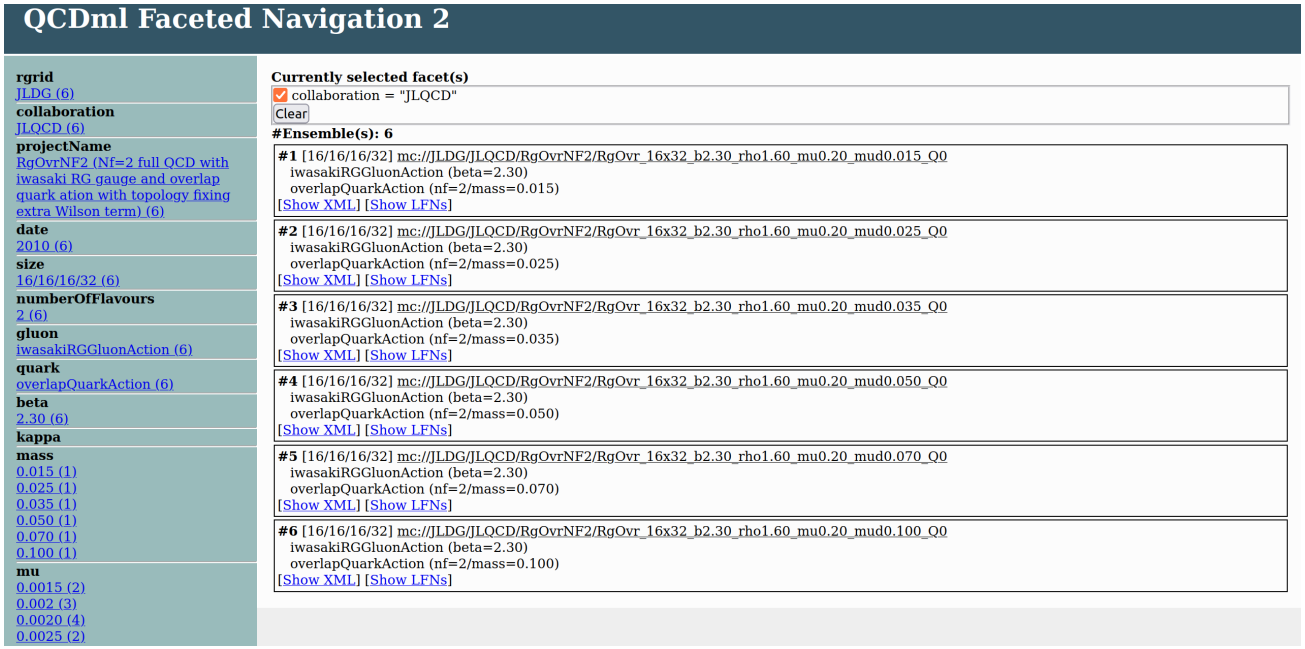
Perl scripts that invoke the right endpoints on the different services, construct the appropriate CURL commands and show the raw response


```
curl -k -L -s -S -X GET -H 'Content-Type: application/json' -H 'accept: */*'
     https://idefix-vm10.zeuthen.desy.de/ildg/mdc/config?id=lfn://ldg/etmc/
     tmqcd_nf2/tlSym_b4.2_L48T96.154073_mu0.002/conf.0000
```

Examples

FacetNavi

Web service that crawls the database and presents a Faceted Navigation system based on selected fields of the MDC schema



FacetNavi Homepage

Examples

OAI-PMH

An interface that allows automatic metadata harvesting following the Open Archives Initiative (OAI) standard protocol. It exposes the metadata in QCDmlEnsemble, Dublin Core and DataCite formats.

Example: list ensembes in DataCite format

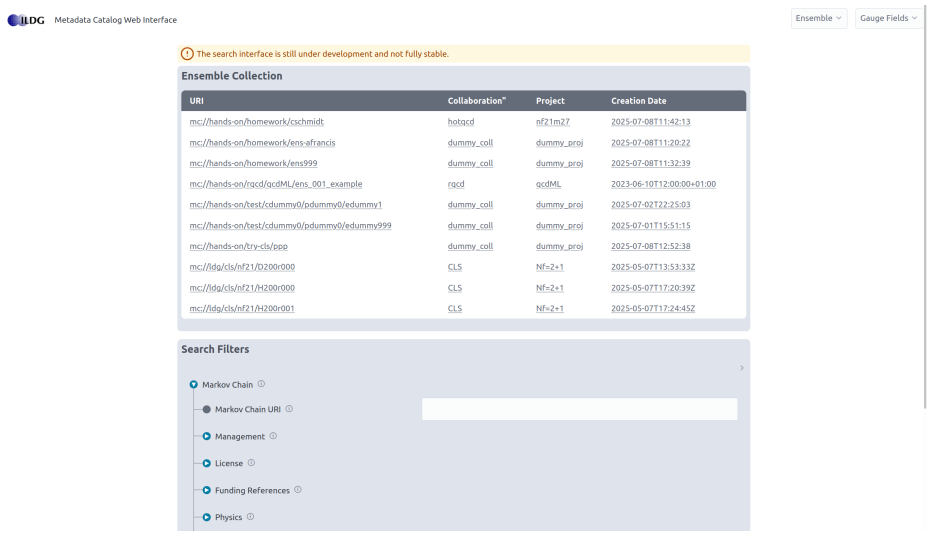
This will be more useful once an ensemble publishing procedure is set up

Examples

PUNCH4NFDI Metadata GUI

Web service that dynamically generates a form based on an MDC schema and allows users to search through catalogs and (hopefully soon) generate XML documents more easily.

The same form can be used for **search** and **markup** of XML documents.



GUI Homepage

Because of the requirements of PUNCH4NFDI, this is generic, i.e. it is not coupled to a specific schema. However, it assumes an ILDG-like API

What is a REST API?

- **REST** stands for *Representational State Transfer*.
- A REST API allows communication between client and server over HTTP.
- It uses standard HTTP methods to perform operations on resources.
- Each resource is identified by a unique URL.
- REST is stateless: each request from client to server must contain all information to understand and process the request.

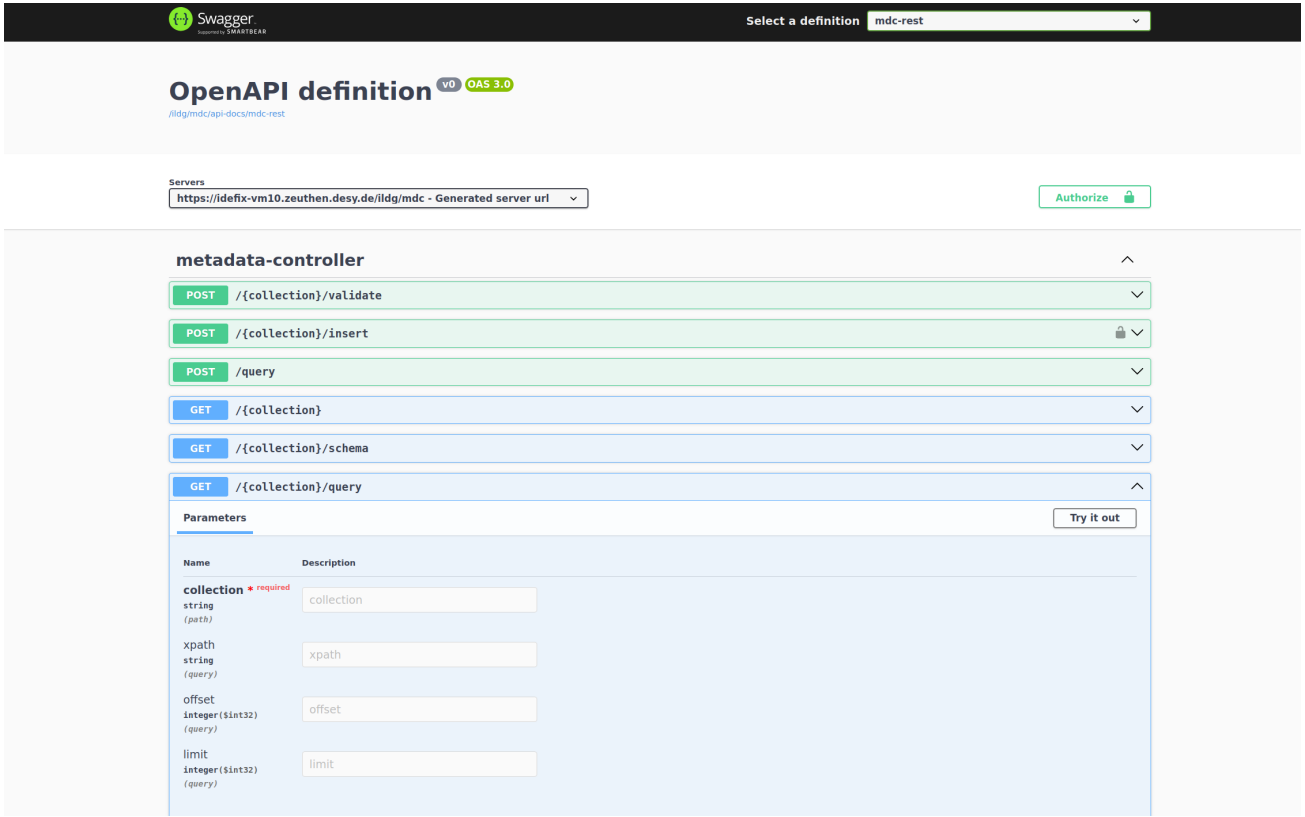
Common HTTP Methods in REST

- **GET**: Retrieve data from the server.
  - e.g., `GET /mdc/config?id=lfn://my-lfn`
  - e.g., `GET /mdc/config/query?mc=mcu://my-mcu`
- **POST**: Send new data to the server. The content is sent in the HTTP request body and has to be marked by the `Content-Type:` attribute
  - e.g., `POST /mdc/config/validate`
  - e.g., `POST /mdc/config/insert*`
- **PUT**: Update existing data.
  - e.g., `PUT /mdc/config/update*`
- **DELETE**: Remove data from the server.
  - e.g., `DELETE /mdc/config/delete?id=lfn://my-lfn*`

The Swagger UI

- Swagger is an open-source framework for designing, building, documenting, and consuming RESTful web services.
- It uses a standardized interface description format known as the OpenAPI Specification (OAS).
- Helps developers understand and interact with APIs without access to source code or documentation.
- Swagger UI is a tool that automatically generates a web-based user interface from an OpenAPI (Swagger) specification.
- Allows users to visualize and interact with the API’s endpoints.
- Supports testing API operations directly from the browser.

The Swagger UI



LDG Swagger UI page

The MDC API

method	URL	response type
GET	/mdc/info	JSON
GET	/mdc/properties	JSON
GET	/mdc/ensemble/schema	JSON/XML
GET	/mdc/config/schema	JSON/XML
POST	/mdc/ensemble/validate	JSON
POST	/mdc/config/validate	JSON
GET	/mdc/ensemble?id=$\langle MCU \rangle$	JSON/XML
GET	/mdc/config?id=$\langle LFN \rangle$	JSON/XML
GET	/mdc/ensemble/query?xpath=$\langle xp \rangle$	JSON
GET	/mdc/config/query?mc=$\langle MCU \rangle$	JSON

All requests should have an `Accept: application/json` header (or occasionally `application/xml`)

The FC API

method	URL	response type
GET	/fc/info	JSON
GET	/fc/properties	JSON
GET	/fc/list?lfn=$\langle LFN \rangle$	JSON
GET	/fc/list?surl=$\langle SURL \rangle$	JSON

JSON Responses

```
{
    "status": "SUCCESS",
    "info": {...} // optional info if SUCCESS
    "result": {
        // the content of the request
    }
}
```

JSON Error Response Example

```
{
    "status": "DUPLICATE_RECORD",
    "info": {
        "code": 400,
        "message": "Id=mc://a/b/c already exists",
        "description": [ "..." ],
        "timestamp": 1673550000,
        ...
    },
    "result": null
}
```

The MDC basic endpoints

- `/info` returns information about the regional grid
- `/properties` returns information and implementation details about the MDC server configuration, which collections are present and which search options are enabled
- `/mdc/<collection>/schema` returns the XSD file for either the ensemble or configurations
- `/mdc/<collection>/validate` is a POST request with a header `Content-Type: application/xml` and content a plain XML file. The server then validates the document against the collection schema and returns the result

/mdc/<collection>?id=<ID>

```
{
    "status": "SUCCESS",
    "info": {...} // optional info if SUCCESS
    "result": {
        "xml":"<?xml version=\"1.0\">....",
        "created": ..., // optional timestamp
        "updated": ..., // optional timestamp
        "aca": ..., // optional access control attributes
        "tags": ... // optional list for tagging
    }
}
```

/mdc/<collection>/query?xpath=<xp>

The parameter <xp> in the URL has to be a valid XPATH expression, which will then be evaluated on the whole collection. The results are a list of unique IDs that match the query

```
{
    "status":"SUCCESS",
    "offset":0,  // start idx in the results list
    "limit":100, // max N of results to show
    "count":2,   // number of results shown in this response
    "total":2,   // total number of results
    "result":[ "mc://a/b/c", "mc://x/y/z", ... ]
}
```

/mdc/query

There is also a POST query endpoint (not-standard). This takes a JSON request body and enables more powerful queries filtering

```
{
    "queryBody": {
      "collection": ...,
      "aca": ...,
      "tags": [...],
      "xpath": {
        "query": "...", // the query string
      }
    },
    "outputFilter": {
      "offset": 0,
      "limit": 100,
      "direction": ..., // sorting direction
      "sortBy": ...,    // a field to sort along by
      "fields": [...]  // list of fields to show for each entry
    }
  }
```

/mdc/query

For example:

```
{
    "queryBody": {
        "collection": "ensemble",
        "xpath": {
            "query": "/markovChain/management/collaboration[./text()='etmc']"
        }
    },
    "outputFilter": {
        "limit": 10,
        "sordBy": "created",
        "fields": ["xml"]
    }
}
```

A note on Quick Search

- Since searching via XPATH is slow on the database, the reference implementation supports *Quick Search Fields*.
- These a set of fields in the metadata schema that are extracted when the XML document is uploaded and stored in separate columns in the database.
- The keys are listed in the `/properties` endpoint, chosen when deploying the catalog

→ much faster search on commonly used fields for ensembles
→ prevents database slowdown when searching through configurations

Usage: `/mdc/<collection>/query?<qkey>=<value>`
Example: `/mdc/config/query?mc=<MCU>`

A second look at FacetNavi

FacetNavi makes periodic requests to the MDC to get a list of all ensembles on all grids (a simple query without any XPATH).

Then it extracts a set of predetermined fields from the XML (grid, collaboration, project, date, number of flavors, action, beta, mass) and stores them into a **local database table**.

The webpages are then generated automatically by the data in the columns of the table, and each switch is a simple SELECT on the database

A second look at the Web Interface

The MDC Web Interface has to be completely agnostic of the content of the catalogs. This is only possible with a standard API across all catalogs in all regional grids

Initialization:

- Get info from MDC about collection and XML schema files
- Parse the XML schema and construct a dynamic web form out of it

Search:

- Read user form data
- construct proper XPATH query
- construct proper POST request
- parse HTTP response and build results table

Markup:

- Read user form data
- Assemble an XML document and return it to the user
- Validate the XML against the MDC schema