# DLCM Integration Guide

# Table of Contents

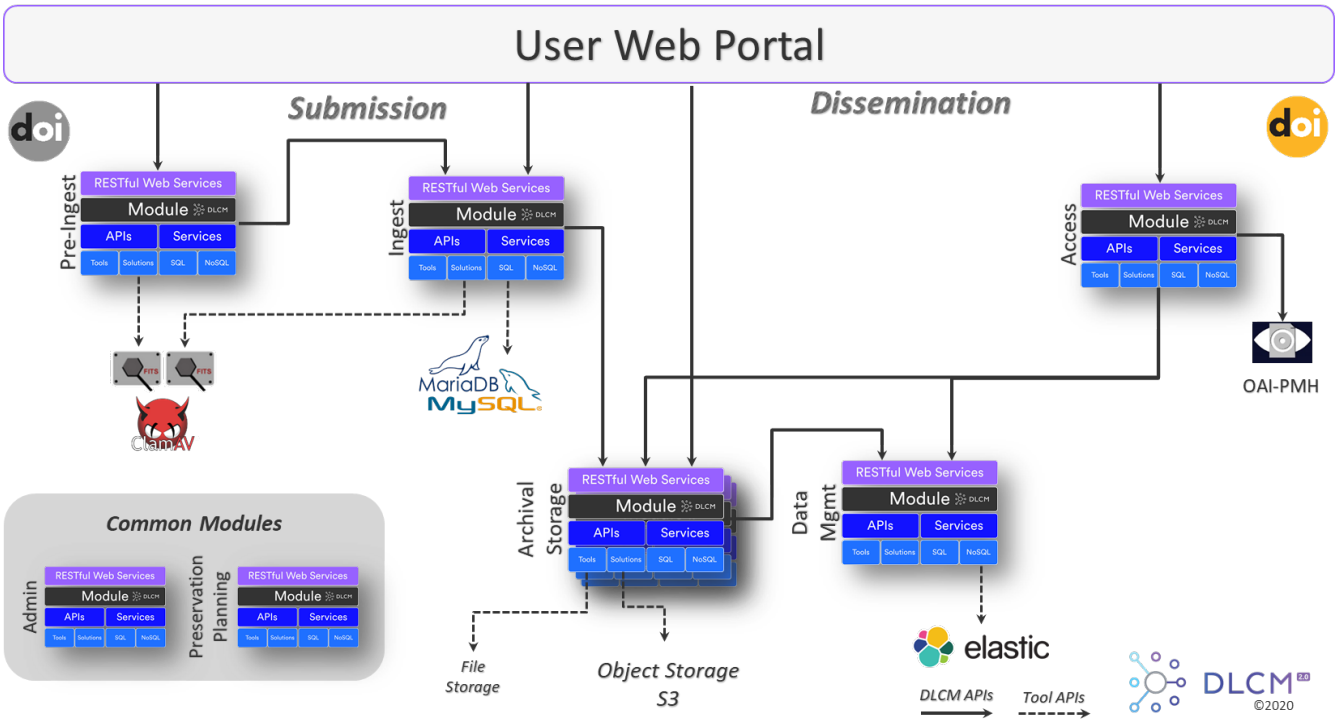
DLCM Solution **v3.1.5**, *2026-02-09*


The current documentation is available in HTML or PDF.

# Chapter 1. DLCM Architecture

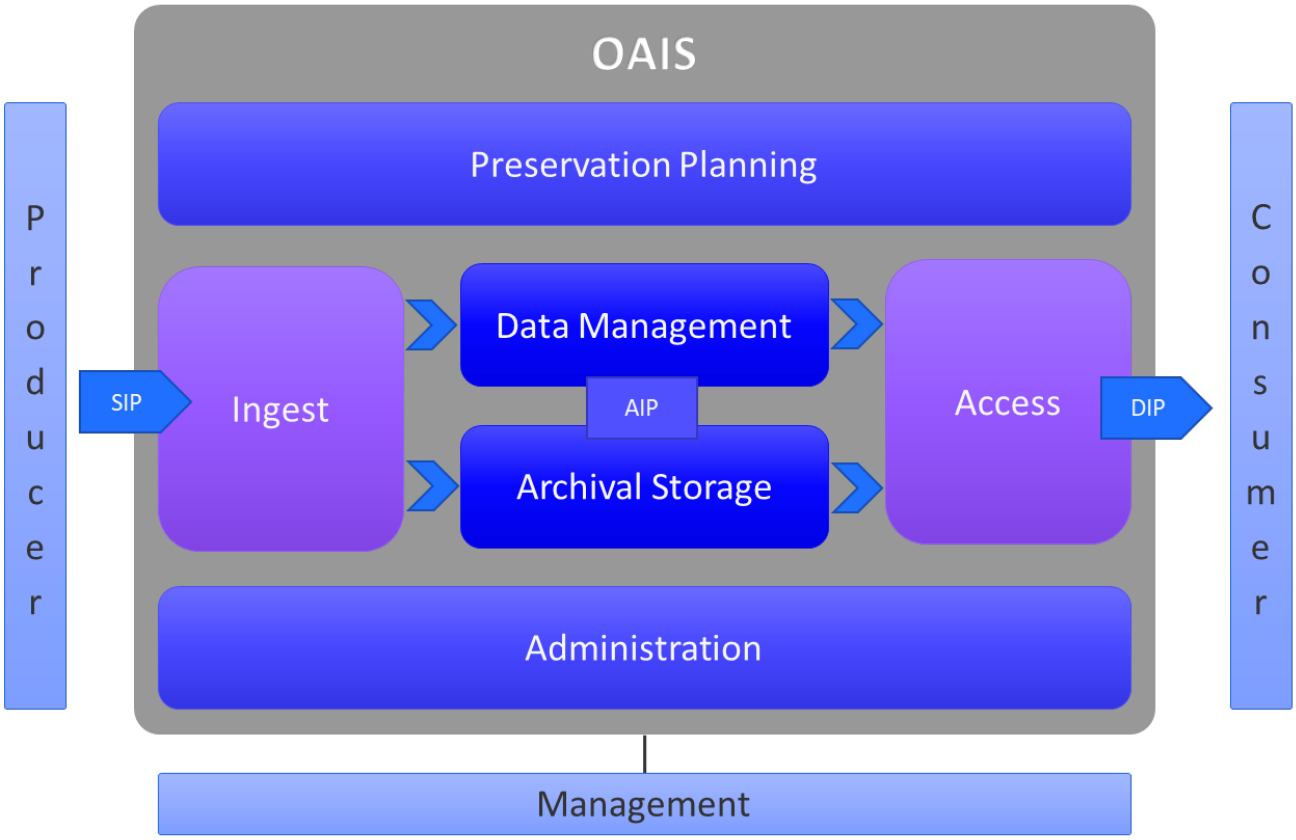
The DLCM solution design is compliant with the OAIS model and follows current best practices of preservation. The solution architecture is open, flexible and modular so as to be scalable, sustainable, and to facilitate its integration with other information systems. How such integrations can be performed constitutes the topic of this document.

## 1.1. DLCM Solution




## 1.2. OAIS Model

**O**pen **A**rchival **I**nformation **S**ystem
**ISO 14721**



# Chapter 2. Integration Points

## 2.1. For Submission

There are three ways to deposit data files into the DLCM system:

1. By submitting individual data files
2. By using a package containing one or several data files
3. Based on a SIP (Submission Information Package)

See the details in Submission Integration section.

## 2.2. For Dissemination

Once the data files have been submitted and archived, the research community can access them:

1. By getting directly an archive with its ID
2. By searching on archive metadata
3. By exporting the AIP (Archival Information Package) through a DIP (Dissemination Information Package)
4. By exporting metadata with OAI-PMH protocol

See the details in Dissemination Integration section.

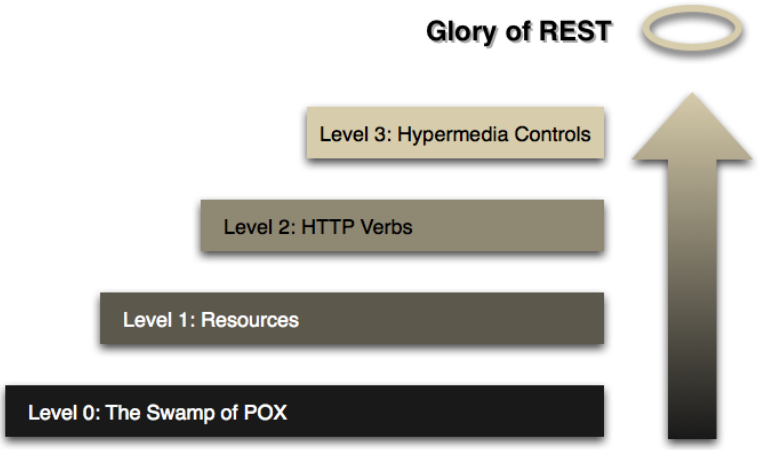
## 2.3. For Developers

- All web services are detailed in API Documentation.
- The API are available in OpenAPI format. See OpenAPI Tools. The definition is available in link:
  - DLCM OpenAPI Specification v3.0
  - DLCM OpenAPI Specification v3.1
- The DLCM tools is a batch tool. The documentation is available in DLCM Tools Documentation.

# Chapter 3. REST Web Services

## 3.1. Overview

The DLCM APIs are RESTful web services based on the best practices. The implementation corresponds to the third level of Leonard Richardson's Maturity Model:

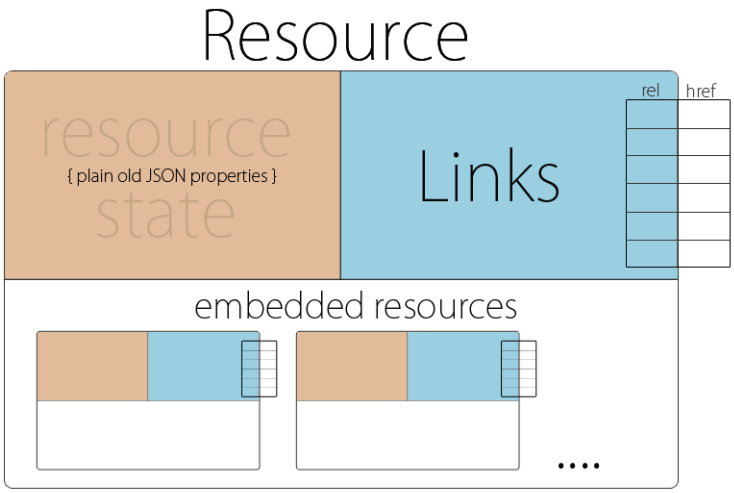



*Source* : (crummy.com, 2008)

More details about these concepts are available on the following links:

- https://spring.io/guides/tutorials/bookmarks/
- https://martinfowler.com/articles/richardsonMaturityModel.html

The data format of the web service is JSON,with HATEOAS & HAL support:




*Source* : (stateless.co, 2011)

### 3.1.1. URL Structure

The URL of each REST resource is constructed according to the following rule:

`http(s)://<root context>/<module>/<things>`

Where:

- `http(s)` is the protocol which can be secured depending on the installation configuration.
- `<root context>` is the root context of the application, defined in the configuration.
- `<module>` is the functional module (see DLCM Architecture): the different module names are detailed in the DLCM Modules section in the Annexes.
- `<things>` is the name of the REST resource: it must be a **noun in plural form**.

The naming convention, applied only for `<things>`, respects the camel case syntax, with a lower case character for the first one.

There are some examples of root contexts in the *demo* environment

### 3.1.2. CRUD Operations

By default, for each REST resource, the CRUD actions are available like this:

| HTTP verb | CRUD action | Collection | Instance |
|---|---|---|---|
| `POST` | Create<br>*Used to create a new resource* | □ | □ |
| `GET` | Read<br>*Used to retrieve a resource or resource list* | □ | □ |
| `PATCH` | Update *No creation*<br>*Used to update an existing resource, including partial updates* | □ | □ |
| `DELETE` | Delete<br>*Used to delete an existing resource* | □ | □ |

The HTTP verb for an action on a resource is `POST`:
`http(s)://<root context>/<module>/<things>/<thingID>/<action>`.

### 3.1.3. HTTP Status Codes

RESTful notes tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP status codes.

| Status code | Usage |
|---|---|
| `200 OK` | The request completed successfully |
| `201 Created` | A new resource has been created successfully. The resource's URI is available from the response's `Location` header |
| `204 No Content` | An update to an existing resource has been applied successfully |
| `400 Bad Request` | The request was malformed. The response body will include an error providing further information |

| Status code | Usage |
|---|---|
| 401 Unauthorized | Authentication is required to access to this resource |
| 403 Forbidden | You are not allowed to access to this method for this resource |
| 404 Not Found | The requested resource did not exist |
| 405 Method Not Allowed | The requested method is not supported for this resource |

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

### 3.1.4. Error Details

```
{
    "path": "http(s)://<root context>/<module>/<things>",
    "status": "BAD_REQUEST",
    "error": "Type of error",
    "message": "Message to explain the issue",
    "timeStamp": "DDD MMM YY hh:mm:ss CEST YYYY",
    "statusCode": 400
}
```

Contains the malformed request information, which describes the problem on the request:

- The ***path*** field is the url of the resource concerned by the problem.
- The ***status*** field is the status of the request (always 'BAD_REQUEST' in this case).
- The ***error*** field is the error that occurs on the request.
- The ***message*** field is the message that details the problem.
- The ***timeStamp*** field is the time at which the error occurred.
- The ***statusCode*** field is the status code of the request (always '400' in this case) .

In the case in which a body object is provided, the ***validationErrors*** field is also added to the fields above. The value of this field is an array that contains for each malformed field:

- The ***fieldName*** field that contains the name of the malformed field.
- The ***errorMessages*** field array that contains the list of errors in this field.

Example of a deposit submission with a malformed body = {}

```
{
    "path": "http(s)://<root context>/<module>/<things>",
    "status": "BAD_REQUEST",
    "error": "None",
    "message": "Validation failed",
    "timeStamp": "Fri May 17 11:39:15 CEST 2019",
    "validationErrors": [
        {
```

```
            "fieldName": "title",
            "errorMessages": [
                "can't be null"
            ]
        },
        {
            "fieldName": "description",
            "errorMessages": [
                "can't be null"
            ]
        },
        {
            "fieldName": "organizationalUnitId",
            "errorMessages": [
                "can't be null"
            ]
        }
    ],
    "statusCode": 400
}
```

Example of a malformed deposit submission with no body

```
{
    "path": "http(s)://<root context>/<module>/<things>",
    "status": "BAD_REQUEST",
    "error": "Required request body is missing: ...",
    "message": "Request not readable",
    "timeStamp": "Fri May 17 12:53:29 CEST 2019",
    "statusCode": 400
}
```

Example of a malformed deposit submission with a body = []

```
{
    "path": "http(s)://<root context>/<module>/<things>",
    "status": "BAD_REQUEST",
    "error": "JSON parse error: ...",
    "message": "Request not readable",
    "timeStamp": "Fri May 17 13:04:39 CEST 2019",
    "statusCode": 400
}
```

## 3.2. Collection

A collection of REST resources is a list of JSON objects. The list has its own structure, is paginated,

filterable and sortable.

The *collection* URL is:

`http(s)://<root context>/<module>/<things>`.

### 3.2.1. Structure

```
{
  "_data" : [
    { "object" : "#1" },
    { "object" : "#2" },
    { "object" : "#3" },
    { "object" : "#4" }
  ],
  "_page": {
    "currentPage" : 0,
    "sizePage" : 20,
    "totalPages" : 1,
    "totalItems" : 4
  },
  "_links" :  {
    "self" : {
      "href" : "URL of the collection"
    },
    "module" : {
      "href" : "URL of the DLCM module"
    }
  }
}
```

#### Data Section

The *Data* section contains an array of JSON representations, corresponding to business objects (i.e. *things*). The details of these objects can be found in the technical documentation (i.e. API Documentation) provided with the DLCM solution.

#### Page Section

The *Page* section contains the pagination information, which describes the current position:

- The ***currentPage*** field is the page number of the current page: it starts at 0.
- The ***sizePage*** field is the size of each page: the default is set to 20, the max value is 2000.
- The ***totalPages*** field is the total number of pages for the current page size.
- The ***totalItems*** field is the total number of objects for the current selection.

#### Links Section

The *Links* section contains the links corresponding to the current collection. This list is dynamic and depends on the state of the collection:

- The ***self*** link is the current URL: it is *always* present.
- The ***module*** link is the URL to access the current module.
- The ***next*** link is the URL to go to the next page, available only if it exists.
- The ***previous*** link is the URL to go to the previous page, available only if it exists.
- The ***lastCreated*** link is the URL to get the list sorted by creation date in descending order.
- The ***lastUpdated*** link is the URL to get the list sorted by last update date in descending order.
- Some other links could be available depending on the current resource: these links are detailed in the API documentation of the resource.

Example of institution list


```
{
  _data : [
    {
      resId :  "7f9df7bb-5eab-4823-98a0-abb668731de5",
      name : "UNIGE",
      description : "Université de Genève",
    },
    {
      resId : "18284eb1-de0b-427e-9e8c-c541cb35e818",
      name : "EPFL",
      description : "Ecole Polytechnique Fédérale de Lausanne",
    },
    {
      resId : "e8a9b74d-7b84-4958-be62-9b0b1d83a360",
      name : "ETH",
      description : "ETH Zürich",
    }
  ],
  _page : {
    currentPage : 0,
    sizePage : 20,
    totalPages: 1,
    totalItems: 4
  },
  _links: {
    self : {
      href : "http://localhost:16105/dlcm/admin/institutions"
    },
    module : {
      href :  "http://localhost:16105/dlcm/admin"
    },
```

```
    lastCreated : {
      href : "http://localhost:16105/dlcm/admin/institutions?sort=creation.when,desc"
    },
    lastUpdated : {
      href :
"http://localhost:16105/dlcm/admin/institutions?sort=lastUpdate.when,desc"
    }
  }
}
```

### 3.2.2. Usage

#### To get a list of things

The different parameters can be used individually or together.

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/<module>/<things>` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `size=<page size>` | The page size |
| | `page=<page number>` | The current page number |
| | `<field name>=<field value>` | To apply a filter on a field if the field is embedded in a sub structure, the field name must be fully named with "." for each level:+ `<sub structure name>.<field name>` |
| | `sort=<field name>[,desc]` | To sort a field<br>By default, the sort is ascending. desc option permits to have descending order. |
| **Expected Return Code** | `200` | *Success* |
| **Return Object** | JSON Collection object | See Structure |

Examples

1. To filter by creation date:
   `http(s)://<root context>/<module>/<things>?sort=creation.when`
2. To sort by most recent objects:
   `http(s)://<root context>/<module>/<things>?sort=creation.when,desc`
3. To get page 10 composed of 5 elements:
   `http(s)://<root context>/<module>/<things>?page=10&size=5`

## 3.3. Instance

The instance of REST resource is the instance of an object with its fields.

The instance URL is:

`http(s)://<root context>/<module>/<things>/<thingID>`.

### 3.3.1. Structure

```
{
  "creation" : {
    "when" : "Creation date & time",
    "who" : "Creation user"
  },
  "lastUpdate" : {
    "when" : "Last update date & time",
    "who" : "Last update user"
  },
  "resId" : "Object ID",
  "fields" : "Object fields...",
  "_links" : {
    "self" : {
      "href" : "URL of the object"
    },
    "list" : {
      "href" : "URL of the object collection"
    },
    "module" : {
      "href" : "URL of the DLCM module"
    },
    "Other link" : {
      "href" :  "Others links of the object"
    }
  }
}
```

The field list elements are:

- The ***creation*** and ***lastUpdate*** fields, containing the information of the action:
  - The ***when*** field is the date and the time, with milliseconds of the action (ex : 2018-03-08T17:42:30.733+0100).
  - The ***who*** field is the user id of the user who has done the action.
- The ***resId*** field is the identifier of the object: it is a UUID.
- Some other fields complete the object description: these fields are detailed in the technical documentation of the resource.

#### Links Section

The *links* section contains a list of links of the object:

- The ***self*** link is the URL of the current object.

- The ***list*** link is the URL pointing to the object collection.
- The ***module*** link is the URL to access the current module.
- Some other links could be available depending on the object: these links are detailed in the technical documentation of the resource.

Example of an institution


```
{
  "creation" : {
    "when" : "2018-03-08T17:42:30.733+0100",
    "who" : "user id of user xxxxxx"
  },
  "lastUpdate" : {
    "when" : "2018-03-08T17:42:30.733+0100",
    "who" : "user id of user yyyyyyy"
  },
  "resId" : "7f9df7bb-5eab-4823-98a0-abb668731de5",
  "name" : "UNIGE",
  "description" : "Université de Genève",
  "_links" : {
    "self" : {
      "href" : "http://localhost:16105/dlcm/admin/institutions/7f9df7bb-5eab-4823-
98a0-abb668731de5"
    },
    "list" : {
      "href" : "http://localhost:16105/dlcm/admin/institutions"
    },
    "module" : {
      "href" : "http://localhost:16105/dlcm/admin"
    },
    "people" : {
      "href" : "http://localhost:16105/dlcm/admin/institutions/7f9df7bb-5eab-4823-
98a0-abb668731de5/people"
    },
    "organizationalUnit" : {
      "href" : "http://localhost:16105/dlcm/admin/institutions/7f9df7bb-5eab-4823-
98a0-abb668731de5/organizationelUnits"
    }
  }
}
```


### 3.3.2. Usage

#### To get a resource

| | |
|---|---|
| **Request** | `http(s)://<root context>/<module>/<things>/<thingID>` |
| **Verb** | GET |

| | | |
|---|---|---|
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Success |
| | `404` | Not found |
| **Return Object** | JSON object | See Structure |

**To create a new resource**

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/<module>/<things>` | |
| **Verb** | POST | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `JSON Object with fields to set` | Object in JSON format. The fields and the structure depend on the type: see API Documentation |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `JSON Object` | See Structure |

**To update a resource**

The resource must already exist.

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/<module>/<things>/<thingID>` | |
| **Verb** | PATCH | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `JSON Object with field to update` | Object in JSON format. The fields and the structure depend on its type: see API Documentation |
| **Expected Return Code** | `200` | Modified |
| | `304` | Not modified |
| | `404` | Not found |
| **Return Object** | `JSON Object with updated fields` | See Structure |

**To delete a resource**

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/<module>/<things>/<thingID>` | |
| **Verb** | DELETE | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |

| | | |
|---|---|---|
| **Expected Return Code** | `200` | Deleted |
| | `404` | Not found |
| | `410` | Gone |
| **Return Object** | `String: "OK"` | If success |

# 3.4. Security

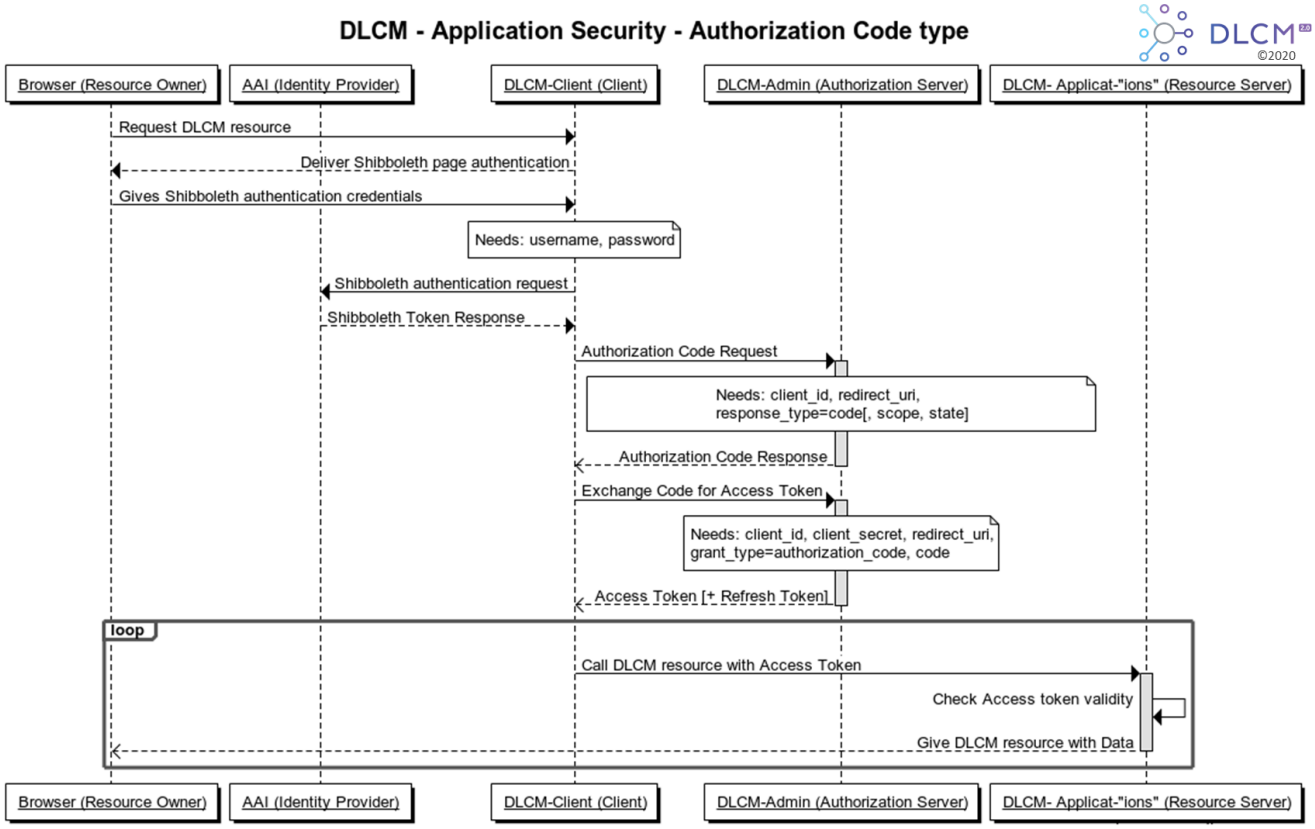
## 3.4.1. Authentication

All web services are secured and require authentication.

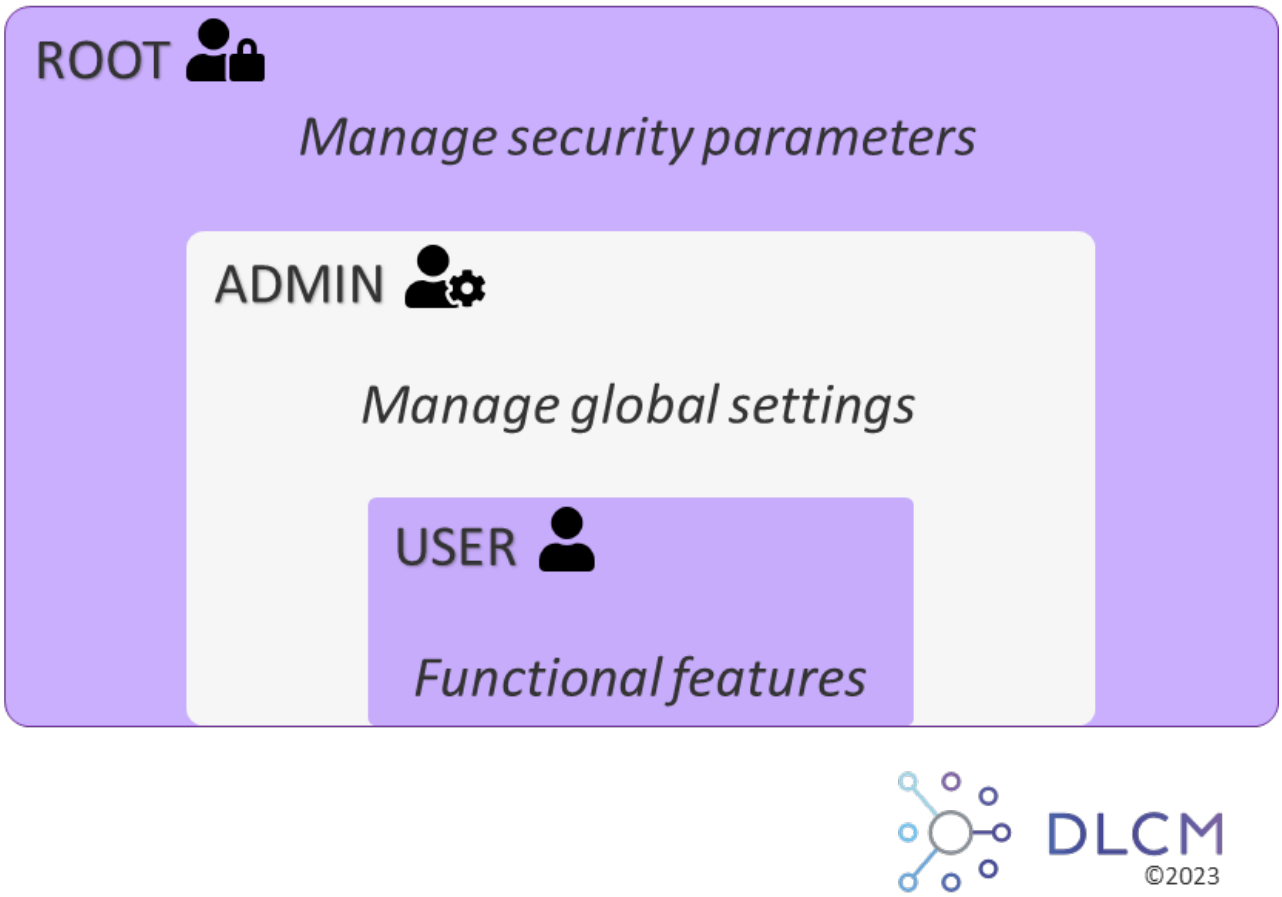
User authentication relies on Switch AAI which is a Single Sign-On (SSO), based on Shibboleth.

Access to Web services relies on OAuth 2.0 access delegation.

OAuth 2.0 is a protocol allowing third-party applications to grant limited access to an HTTP service, either on behalf of a resource or by allowing the third-party application to obtain access on its own. It uses the authorization code grant implementation.




## 3.4.2. Application Roles

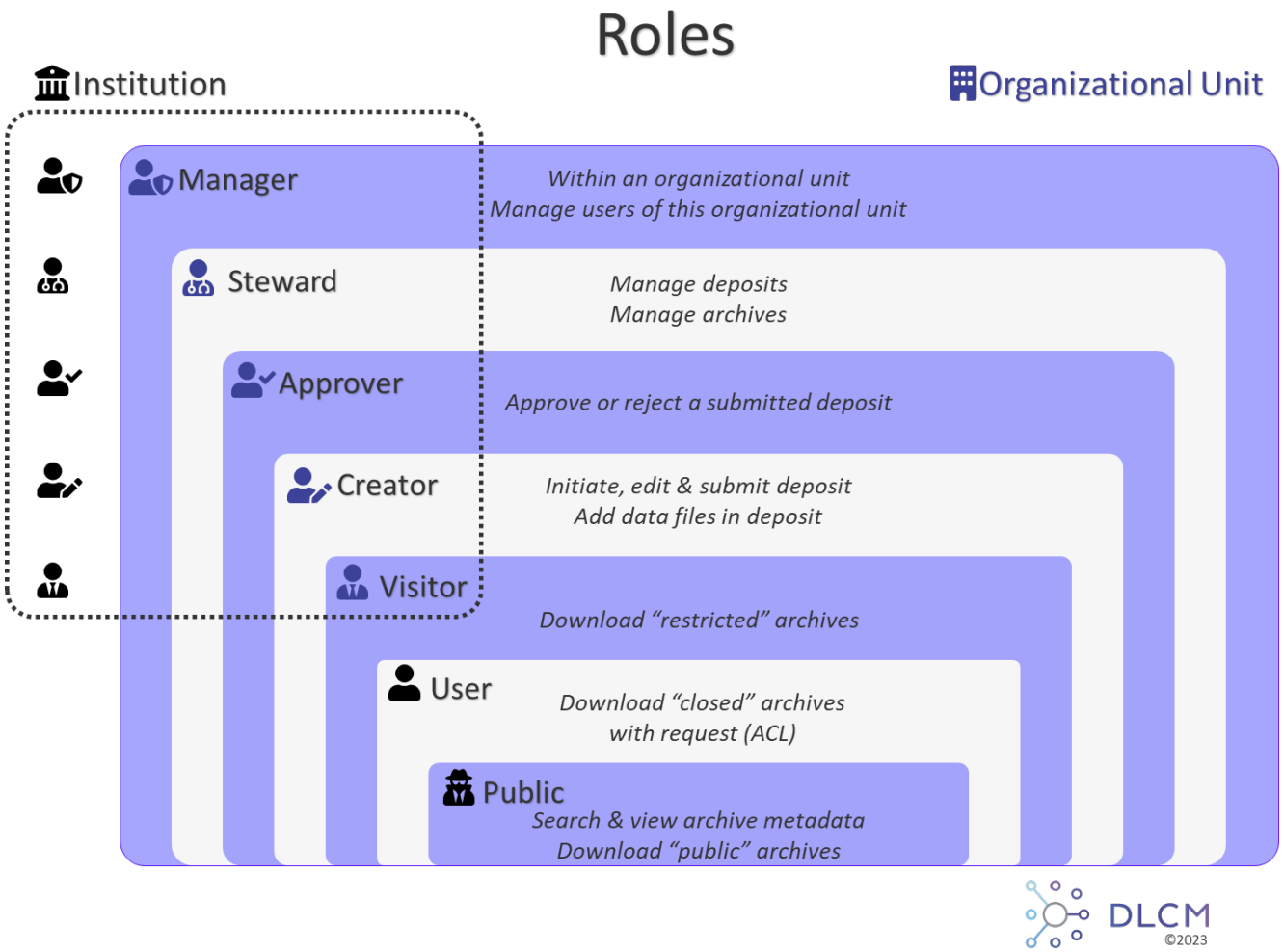



The features are organized in:

- Functional features list for an user
  - Organizational Units, Deposits, Search, etc…
- Global settings list for an administrator
  - Organizational units, People, Institutions, Funding agencies, Submission policies, Preservation policies, Licenses, etc…
- Security parameters list for a root
  - System Configuration, User Roles, Users, etc…

See details in Application Roles.

### 3.4.3. Roles



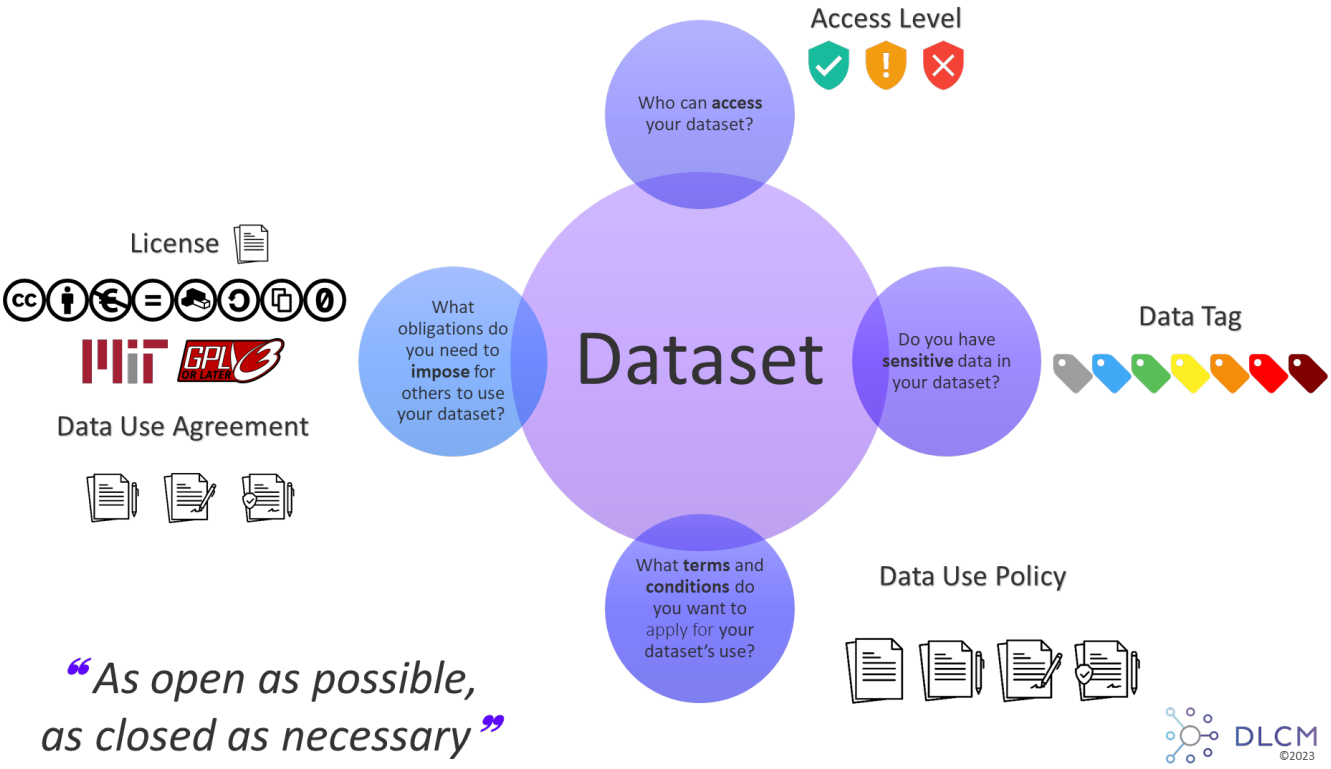

A role can be inherited from an institution if an organizational unit is linked to this institution. For example, if an manager of an institution or an administrator gives the `steward` role to an user, this user will became the steward of all organizational units of this institution.

See details in Roles.

**Organizational Unit Definition**

- An organizational unit is a logical entity where managers can define security rules:
  - Who can submit deposits?
  - Who can download archives?
- Could be a research project, a laboratory, a department or any other organizational group of researchers.

# Chapter 4. Data Access




When data must be archived, at the ingestion, some questions must be answered:

① Who can access the archive? ⇒ Define the correct level of Access Levels

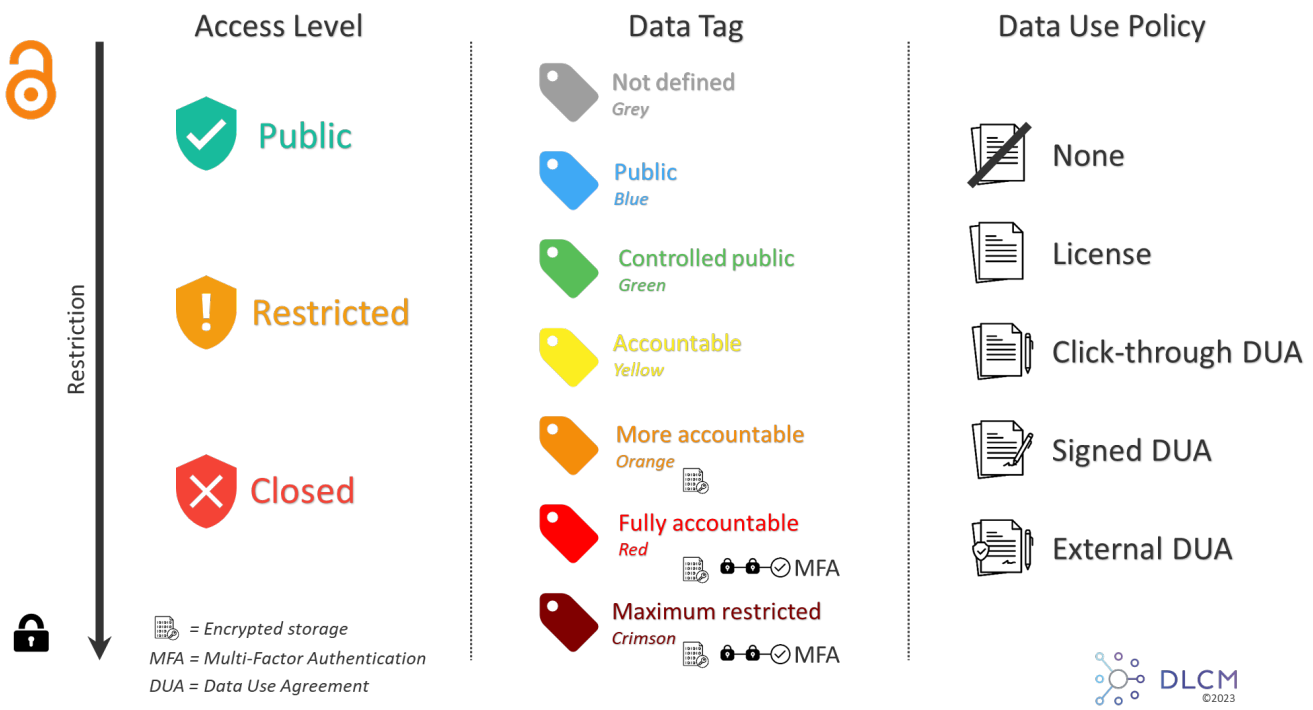
② Are there sensitive data in the archive? ⇒ Define the correct tag of Data Tags

③ What are terms & conditions to use the archive? ⇒ Define the correct policy of Data Use Policies

④ What are obligations to impose for using the archive? ⇒ Choose the *license* or specify the *data use agreement*

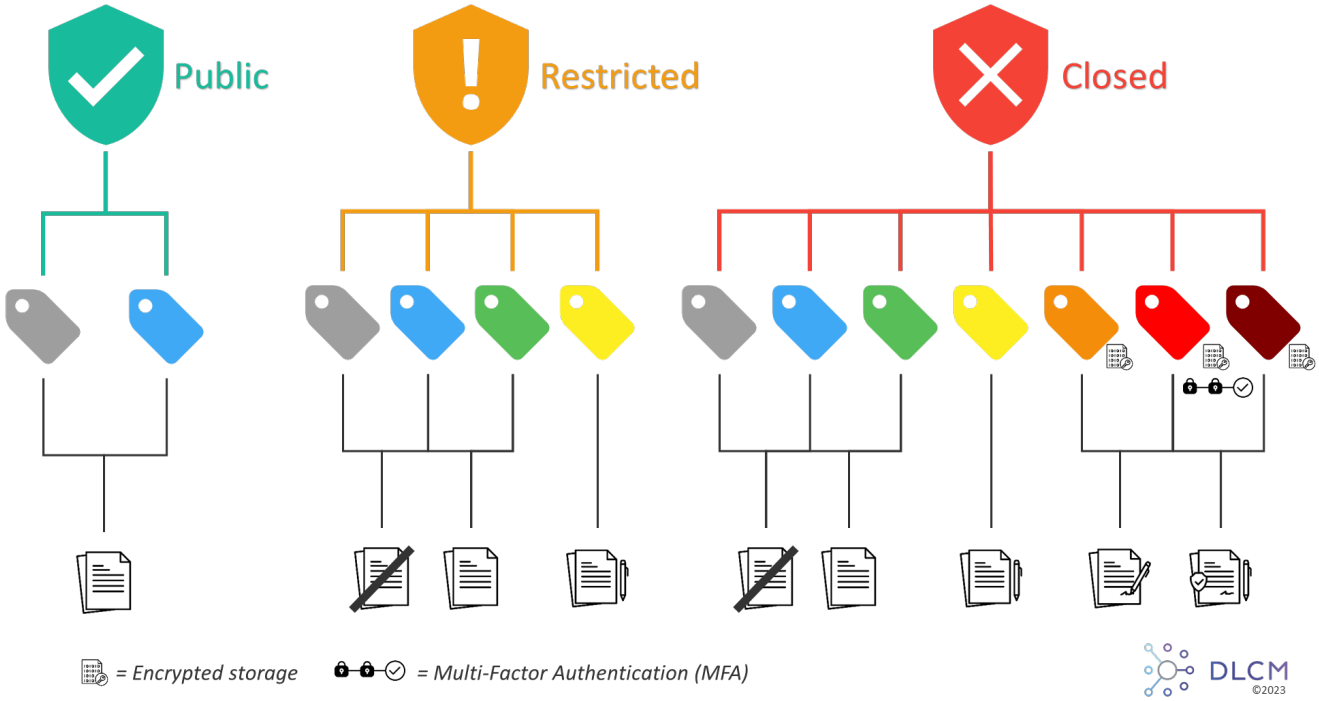
## 4.1. Data Access Scales

Each concept (access level, data tag or data use policy) have a restriction scale to control the access to the archives: from less to more restrictive.



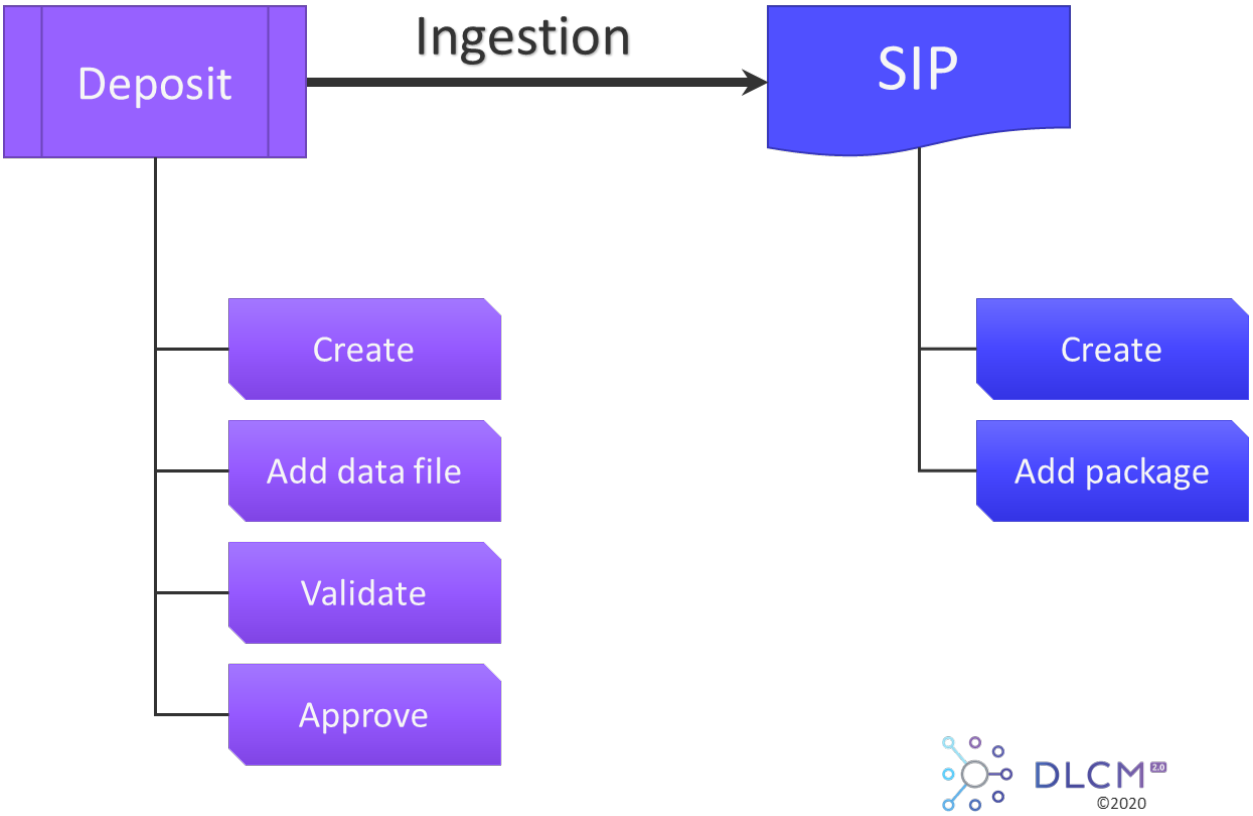

## 4.2. Data Access Compatibility

The compatibility between those concepts are described in the following matrix.



# Chapter 5. Submission Integration

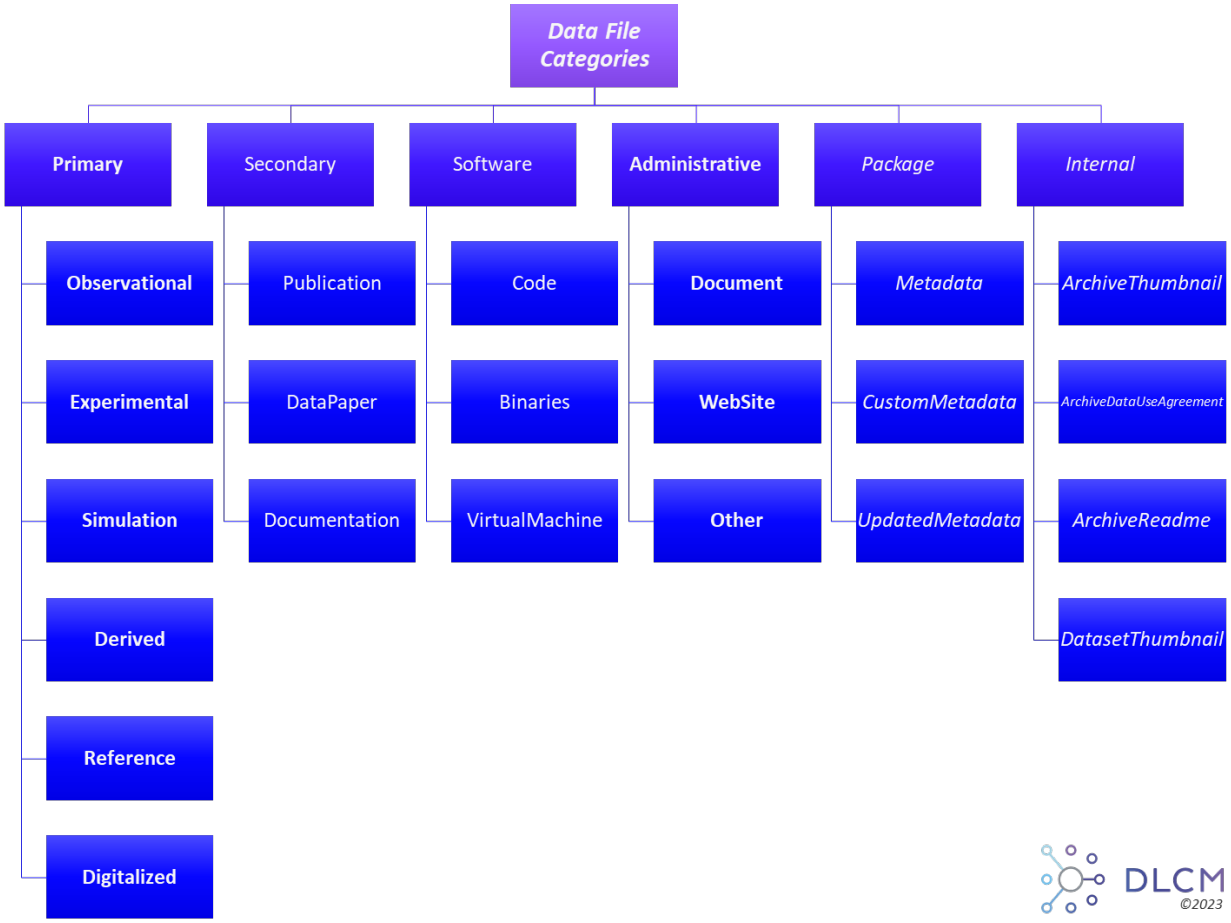
## 5.1. Overview




The ingestion process consists either in the creation of a deposit based on a wizard-like assisted approach, or in using a *ready-to-use* SIP.

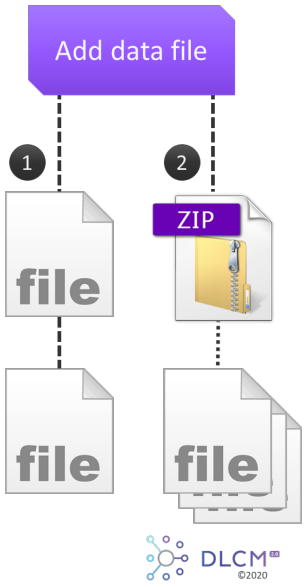
## 5.2. Wizard-like assisted deposit

The deposit operation consists in gathering all data files and the information necessary to create a SIP package. The objective of the wizard is to structure the deposit and to categorize each data file:




The description of each category is detailed at the Data File Categories section in the Annexes.

The data file assignment to a deposit can be done file-by-file (mode ) or by batch (mode ):




### 5.2.1. To create a deposit

| | |
|---|---|
| **Request** | `http(s)://<root context>/preingest/deposits` |
| **Verb** | POST |

| Parameter(s) | *Name* | *Description* |
|---|---|---|
| | `Deposit JSON Object` | See *Deposit* section in API Documentation |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `Deposit JSON Object` | See *Deposit* section in API Documentation |
| **Roles** | Creator (see Roles) | |

Deposit example

The minimal set of information for a deposit is:

```
{
  "organizationalUnitId" : "Organizational unit ID of the data set",
  "title" : "Data set title",
  "year" : 2018,
  "description" : "Data set description"
}
```

### 5.2.2. To deposit data files

To add data files to a deposit, the first option (mode ) is to deposit them one-by-one.

#### By creating an URI

It's possible to provide a URI (useful for large files).

| Request | `http(s)://<root context>/preingest/deposits/<DepositID>/data` | |
|---|---|---|
| **Verb** | POST | |
| **Parameter(s)** | *Name* | *Description* |
| | `Data File JSON Object` | See *Data File* section in API Documentation |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `Data File JSON Object` | See *Data File* section in API Documentation |
| **Roles** | Creator (see Roles) | |

The effective download of the referenced data (see API Documentation) is done asynchronously by the "Pre-Ingest" module, which supports the file (for files on local file systems), http and https protocols.

**By uploading a file**

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/preingest/deposits/<DepositID>/upload` | |
| **Verb** | POST | |
| **Content-type** | `multipart/form-data` | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `file` | Data file to upload |
| | `category` **(*optional*)** | Data file category (see Data File Categories section in the Annexes) |
| | `type` **(*optional*)** | Data file sub-category (see Data File Categories section in the Annexes) |
| | `folder` **(*optional*)** | Sub-folders of data file |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `Data File JSON Object` | See *Data File* section in API Documentation |
| **Roles** | Creator (see Roles) | |

If the data file is the descriptive metadata of the dataset, it must respect the deposit metadata schema. If not, the data file will have a status `In-Error`.

### 5.2.3. To deposit a data files package

The second option (mode ) is to add data files in a deposit by batches. The batch mode supports zip files, containing all the data files to upload, including sub-folders.

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/preingest/deposits/<DepositID>/upload-archive` | |
| **Verb** | POST | |
| **Content-type** | `multipart/form-data` | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `file` | Zip file which contains data files |
| | `category` **(*optional*)** | Data file category (see Data File Categories section in the Annexes) |
| | `type` **(*optional*)** | Data file sub-category (see Data File Categories section in the Annexes) |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `Array of Data file JSON Object` | See *Data File* section in API Documentation |
| **Roles** | Creator (see Roles) | |

### 5.2.4. To get the deposit metadata schema

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/preingest/deposit/schema` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `Descriptive Metadata XML schema` | XML schema file |
| **Roles** | All (see Roles) | |

### 5.2.5. To submit a deposit for approval

This step is optional. It depends of submission policy if an approval is expected.

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/preingest/deposits/<DepositID>/submit-for-approval` | |
| **Verb** | POST | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Modified |
| | `304` | Not modified |
| | `404` | Not found |
| **Return Object** | `Result JSON` | Action result<br>`{`<br>`"message": "Deposit status changed successfully",`<br>`"resId": "<DepositID>",`<br>`"status": "EXECUTED"`<br>`}` |
| **Roles** | Creator (see Roles) | |

### 5.2.6. To approve a deposit

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/preingest/deposits/<DepositID>/approve` | |
| **Verb** | POST | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |

| | | |
|---|---|---|
| **Expected Return Code** | `200` | Modified |
| | `304` | Not modified |
| | `404` | Not found |
| **Return Object** | `Result JSON` | Action result<br>`{ "message": "Deposit status changed successfully", "resId": "<DepositID>", "status": "EXECUTED" }` |
| **Roles** | Approver (see Roles) | |

# 5.3. By using a SIP

## 5.3.1. To create a SIP

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/ingest/sip` | |
| **Verb** | POST | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `SIP JSON Object` | See *SIP* section in API Documentation |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `SIP JSON Object` | See *SIP* section in API Documentation |
| **Roles** | Creator (see Roles) | |

SIP example

The minimal set of information for an SIP is:

```
{
  "info" : {
    organizationalUnitId" : "Organizational unit ID of the SIP",
    "name" : "Name of the SIP",
    "description" : "Description of the SIP"
  }
}
```

## 5.3.2. To submit a SIP package

| | |
|---|---|
| **Request** | `http(s)://<root context>/ingest/sip/<sipID>/upload` |
| **Verb** | POST |

| | | |
|---|---|---|
| **Content-type** | `multipart/form-data` | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `Zip file` | The Zip file must contain a metadata XML file and at least one data file. |
| **Expected Return Code** | `201` | Created |
| **Return Object** | `Data File JSON Object` | See *Data File* section in API Documentation |
| **Roles** | Creator (see Roles) | |

The SIP metadata file must be in XML and respect the SIP metadata schema.

### 5.3.3. To get SIP metadata schema

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/ingest/sip/schema` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `SIP DLCM Metadata XML schema` | XML schema file |
| **Roles** | All (see Roles) | |

# Chapter 6. Dissemination Integration

## 6.1. To search archives

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/access/metadata/search?query=<query>` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `Query` | Query criteria |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `Collection JSON Object` | List of Archive information & DataCite metadata (see [archive-example]) with pagination |
| **Roles** | Public (see Roles) | |

Query examples:

- `criterion1 criterion2` ⇒ criterion1 **or** criterion2
- `criterion1 AND criterion2` ⇒ criterion1 **and** criterion2

## 6.2. To get an archive

### 6.2.1. By archive ID

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/access/metadata/<archiveID>` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `Archive Public Metadata JSON Object` | Archive information & DataCite metadata (see [archive-example]) |
| **Roles** | Public (see Roles) | |

Archive public metadata example

```
{
    "resId": "<archiveID>",
    "index": "<index name>",
    "type": "metadata",
    "metadata": {
```

```
        "aip-disposition-approval": "<true/false>",
        "aip-organizational-unit": "<organizational unit ID>",
        "aip-retention": "<retention duration in days",
        "aip-retention-end": "<retention end date>",
        "aip-unit": "<true/false>",
        "aip-size": "<archive size>",
        "creation": "<creation date>"
        "datacite.xml": "<DataCite XML>",
        "aip-container": "BAG_IT",
        "datacite": {
            <DataCite JSON>
        }
    }
}
```

### 6.2.2. By DOI

| **Request** | `http(s)://<root context>/access/metadata/search-doi?doi=<DOI>` | |
|---|---|---|
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `DOI` | DOI to search |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `Archive Public Metadata JSON Object` | Archive information & DataCite metadata (see [archive-example]) |
| **Roles** | Public (see Roles) | |

## 6.3. To download an archive

To download an archive, several steps are needed:

1. To check if a download request exists and to know its status: To get download status
2. To create a download request: To prepare download
3. To download the archive: To download archive content

### 6.3.1. To get download status

| **Request** | `http(s)://<root context>/access/metadata/<archiveID>/download-status` | |
|---|---|---|
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |

| | | |
|---|---|---|
| **Expected Return Code** | `200` | Success |
| | `404` | Not found ⇒ To prepare download |
| **Return Object** | `Download Status` | `SUBMITTED` Download query created<br>`IN_ERROR` Download query in error<br>`IN_PREPARATION` Preparing download query<br>`DOWNLOADING` Downloading AIP<br>`IN_DISSEMINATION_PREPARATION` Preparing DIP<br>`READY` Download query completed ⇒ To download archive content |
| **Roles** | Public (see Roles) | |

### 6.3.2. To prepare download

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/access/metadata/<archiveID>/prepare-download` | |
| **Verb** | POST | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `202` | Accepted |
| **Return Object** | - | |
| **Roles** | Public (see Roles) | |

### 6.3.3. To download archive content

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/access/metadata/<archiveID>/download` | |
| **Verb** | GET | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `None` | - |
| **Expected Return Code** | `200` | Success |
| **Return Object** | `Archive File` | |
| **Roles** | Public (see Roles) | |

## 6.4. To export metadata with OAI-PMH

The OAI-PMH provider of DLCM solution supports version 2.0 of the protocol for metadata harvesting. The specifications are detailed on the Open Archives Initiative website.

| | | |
|---|---|---|
| **Request** | `http(s)://<root context>/access/oai-provider/oai` | |
| **Verb** | GET or POST with content-type application/x-www-form-urlencoded | |
| **Parameter(s)** | ***Name*** | ***Description*** |
| | `OAI parameters` | See OAI-PMH specifications. |
| | `smartView=dlcm_oai2.xsl` | Optional parameter to display OAI XML in a structured way, with XML transformation to generate HTML. |
| **Expected Return Code** | `200` | Success |
| | `503` | Service unavailable, i.e. the Data Management module is not running |
| **Return Object** | `OAI-PMH XML data` | OAI-PMH XML data. See OAI-PMH specifications |
| **Roles** | Public (see Roles) | |

# Chapter 7. Annexes

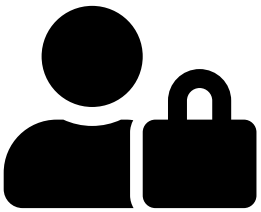
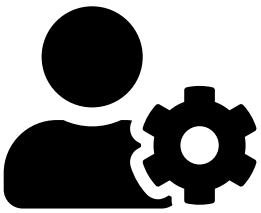
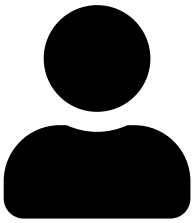
## 7.1. Glossary

| Acronym | Description | Source |
|---|---|---|
| AIC | Archival Information Collection | OAIS |
| AIP | Archival Information Package (i.e. Archive) | OAIS |
| AIU | Archival Information Unit | OAIS |
| API | Application Programming Interface | Software |
| CRUD | Create Read Update Delete | Software |
| Deposit | Research data deposit | DLCM |
| DIP | Dissemination Information Package | OAIS |
| HAL | Hypertext Application Language | Software |
| HATEOAS | Hypermedia As The Engine Of Application State | Software |
| IP | Information Package | OAIS |
| JSON | JavaScript Object Notation | Software |
| OAIS | Open Archival Information System | OAIS |
| REST | REpresentational State Transfer | Software |
| SIP | Submission Information Package | OAIS |
| SOA | Service Oriented Architecture | Software |

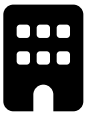
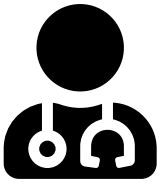
## 7.2. DLCM Modules

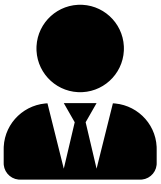
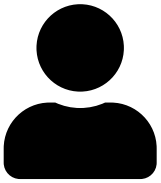
| Module | Description | REST Name |
|---|---|---|
| Pre-Ingest | *Pre-Ingest* module to prepare a deposit in SIP | preingest |
| Ingest | *Ingest* module to check an SIP and to transform it into an AIP | ingest |
| Archival Storage | *Archival Storage* module to check an AIP and to store it | archival-storage |
| Data Mgmt | *Data Management* module to index metadata | data-mgmt |
| Access | *Access* module to manage queries/request and to generate a DIP | access |
| Preservation Planning | *Preservation Planning* module to manage preservation activities | preservation-planning |
| Admin | *Administration* module to manage general settings | admin |

## 7.3. Application Roles

| Icon | Application Role | Description |
|---|---|---|
|  | `ROOT` | A root is the super administrator of the application.<br>⇒ He has access to everything. |
|  | `ADMIN` | An administrator can configure the application by defining new parameters, like *license* or *preservation policy*.<br>⇒ He has access to global settings of the application. |
|  | `USER` | An user is a person who have to use the application.<br>⇒ He has access to the preservation space, based on the permission he has at institution or organizational unit level. |

## 7.4. Roles

| Icon | Role | Within an *institution*  | Within an *organizational unit*  |
|---|---|---|---|
|  | `Manager` | He can manage users of this institution.<br>He can create organizational units for this institution. | He can manage users of this organizational unit,<br>plus the same rights as a *steward*. |
|  | `Steward` | He has the steward role for all organizational units of the institution. | He has the steward role for the organizational unit.<br>He can manage deposits and archives,<br>plus the same rights as an *approver*. |
|  | `Approver` | He has the approver role for all organizational units of the institution. | He has the approver role for the organizational unit.<br>He can approve or reject a submitted deposit, plus the same rights as a *creator*. |

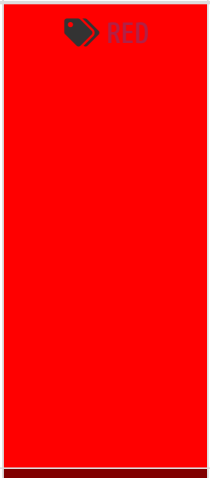
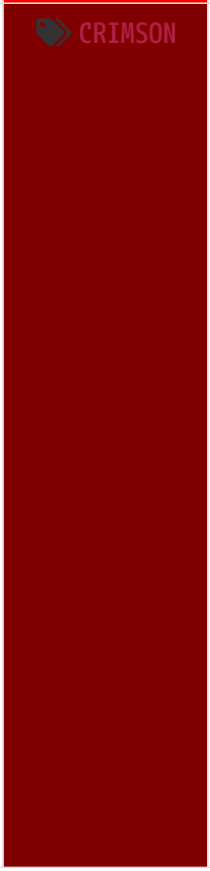
| | | | |
|---|---|---|---|
|  | `Creator` | He has the creator role for all organizational units of the institution. | He has the creator role for the organizational unit.<br>He can initiate, edit & submit deposits, add data files in deposits, plus the same rights as a *visitor*. |
|  | `Visitor` | He has the visitor role for all organizational units of the institution. | He has the visitor role for the organizational unit.<br>He can download *restricted* archives,<br>plus the same rights as *user*. |
|  | `User` | A user is an authenticated person (with a login/password) on the application.<br>He has no permission on any institution. | He has no permission on any organizational unit.<br>He can download *closed* archives, if he asked a access request and a *steward* have accepted the request.<br>plus the same rights as *public*. |
|  | `Public` | A public is not authenticated on the application.<br>He has no permission on any institution. | He has no permission on any organizational unit.<br>He can download *public* archives, which are *Open Access*. |

## 7.5. Access Levels

| Icon | Access Level | Description |
|---|---|---|
|  | PUBLIC | The archive is accessible to everyone<br>⇒ Open access. |
|  | RESTRICTED | The archive is accessible to team members (i.e., Org. Unit)<br>⇒ Trusted parties. |
|  | CLOSED | The archive is accessible case by case thank to access control list (ACL)<br>⇒ Individuals. |

## 7.6. Data Tags

| Icon | Tag Type | Description | Transfer | Storage | Access | Requirement Support | Access Level Compatibility |
|---|---|---|---|---|---|---|---|
|  | UNDEFINED | ***Not defined*** Data sensitivity not set to support previous archives. | - | - | - | Supported | All |
|  | BLUE | ***Public*** Non-confidential information, stored and shared freely. | Clear | Clear | Open | Supported | All |
|  | GREEN | ***Controlled public*** Not harmful personal information, shared with some access control. | Clear | Clear | Email, OAuth verified registration | Supported | Restricted and Closed only |
|  | YELLOW | ***Accountable*** Potentially harmful personal information, shared with loosely verified and/or approved recipients. | Encrypted | Clear | Password, Registered , Approval, click-through DUA | Supported | Restricted and Closed only |
|  | ORANGE | ***More accountable*** Sensitive personal information, shared with verified and/or approved recipients under agreement. | Encrypted | Encrypted | Password, Registered, Approval, signed DUA | Supported *If Encryption is enabled on storage device* | Closed only |

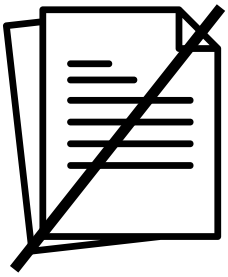
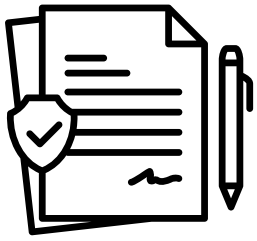
| Icon | Tag Type | Description | Transfer | Storage | Access | Requirement Support | Access Level Compatibility |
|---|---|---|---|---|---|---|---|
|  |   | ***Fully accountable*** Very sensitive personal information, shared with strong verification of approved recipients under signed agreement. | `Encrypted` | `Encrypted` | Two-factor Authentication, Registered, Approval, signed DUA | `Supported` *If* `Encryption` *is enabled on storage device* | `Closed` only |
|  |   | ***Maximum restricted*** Maximum sensitive, explicit permission for each transaction, strong verification of approved recipients under signed agreement. | `Encrypted` | `Multi-encrypted` | Two-factor Authentication, Registered, Approval, signed DUA | `Partially Supported` *If* `Encryption` *is enable on storage device* `Multi-encrypted` *not supported yet* | `Closed` only |

DUA = Data Use Agreement

*Sources*:

- Sharing Sensitive Data with Confidence: The Datatags System

## 7.7. Data Use Policies

| Icon | Data Use Policy | Description |
|---|---|---|
|  | None | No data use policy. No constraint. |
|  | License | A license is mandatory. |
|  | Click-through DUA | A DUA is mandatory. It is stored in the archive and is approved when the user click to request access. |
|  | Signed DUA | A DUA is mandatory. It is stored in the archive. It must be downloaded, signed and provided when the user request access. |
|  | External DUA | A DUA is mandatory. But, it is managed and stored externally, so not in the archive. The control is done outside the application. |

DUA = Data Use Agreement

## 7.8. Deposit Status

| Status | Description |
|---|---|
| APPROVED | The deposit has been approved. |
| CHECKED | The deposit has been checked. |
| CLEANED | The deposit has been cleaned: all date files have been purged. |
| CLEANING | A clean job is currently running. |

| Status | Description |
|---|---|
| COMPLETED | The deposit has been archived. |
| CREATED | The deposit has been created. |
| IN_ERROR | The deposit is in error. An action must be done to fix the deposit. |
| IN_PROGRESS | The deposit is open to add data files. |
| IN_VALIDATION | The deposit must be validated. |
| REJECTED | The validation process has rejected the deposit. |
| SUBMITTED | The deposit process is started. |

## 7.9. Data File Categories

| Category | *Sub-Category* | Description |
|---|---|---|
| **Primary** | ***Primary Data category*** | |
| | *Observational* | Data captured in real-time, usually irreplaceable. For example, sensor data, survey data, sample data, neuro-images. |
| | *Experimental* | Data from lab equipment, often reproducible, but can be expensive to reproduce. For example, gene sequences, chromatograms, and toroid magnetic field data. |
| | *Simulation* | Data generated from test models. The model and its parameters are as important as the result and are also part of the data. For example, climate models, economic models, etc. |
| | *Derived* | Data derived from other data. Generally reproducible but can also be expensive (for instance parameters of LLM). For example, text and data mining, compiled database, 3D models, etc. |
| | *Reference* | A (static or organic) conglomeration or collection of smaller (peer-reviewed) datasets, most probably published and curated. For example, gene sequence databanks, chemical structures, or spatial data portals. It can be expensive to create. |
| | *Digitalized* | Digital version of analogue objects. For example, manuscripts, books, audio etc. |
| **Secondary** | ***Secondary Data category*** | |

| Category | *Sub-Category* | Description |
|---|---|---|
| | *Publication* | Research publication or article |
| | *DataPaper* | Research data paper |
| | *Documentation* | Other documentation |
| **Software** | ***Software category*** | |
| | *Code* | Code or programs |
| | *Binaries* | Binaries or executables |
| | *VirtualMachine* | Images of virtual machines |
| **Administrative** | ***Administrative category*** | |
| | *Document* | All kinds of documents |
| | *WebSite* | Web sites |
| | *Other* | Others types of files |
| **Package** | ***DLCM Package category*** | |
| | *InformationPackage* | DLCM Package (*internal used only*) |
| | *Metadata* | DLCM metadata in XML format |
| | *CustomMetadata* | Specific metadata of a research in JSON or XML format |
| | *UpdatedMetadata* | DLCM updated metadata in XML format |
| | *UpdatePackage* | DLCM Updated Package (*internal used only*) |
| **Internal** | ***DLCM Archive category*** | |
| | *DatasetThumbnail* | Dataset Thumbnail (for archive version < 3.1) |
| | *ArchiveThumbnail* | Archive Thumbnail (for archive version >= 3.1) |
| | *ArchiveReadme* | Archive README |
| | *ArchiveDataUseAgreement* | Data Use Agreement (DUA) for archives with sensitive data |

The categories are dependent on the configuration.

- The categories, ***Primary***, ***Secondary*** & ***Software***, are available if research data archiving is enable.
- The category, ***Administrative***, is available if administrative data archiving is enable.

## 7.10. Data File Status

| Status | Description | Action |
|---|---|---|
| CHANGE_DATA_CATEGORY | An update of the data category or of the data type is in progress. | n/a |
| CHANGE_RELATIVE_LOCATION | An update of the relative location is in progress. | n/a |
| CLEANED | The data file is purged after a clean job. | n/a |
| CLEANING | A clean job is currently running. | n/a |
| CREATED | The data file has been created. | n/a |
| EXCLUDED_FILE | The data file has a forbidden format. | This data file must be deleted before submitting deposit. |
| FILE_FORMAT_IDENTIFIED | The file format identification ran successfully. | n/a |
| FILE_FORMAT_SKIPPED | The file format identification was skipped because of the file size. | n/a |
| FILE_FORMAT_UNKNOWN | The file format identification failed. | n/a |
| IGNORED_FILE | The data file has a format which needs a action. | This data file must be deleted or validated before submitting deposit. |
| IN_ERROR | The data file is in error. | An action must be done to fix the data file. |
| PROCESSED | The data file has been processed: downloaded or copied. | n/a |
| READY | The data file processing is completed. | n/a |
| RECEIVED | The data file is created. | n/a |
| TO_PROCESS | The data file is created, before to be processed. | n/a |
| VIRUS_CHECKED | The virus check ran successfully: no virus. | n/a |
| VIRUS_SKIPPED | The virus check was skipped because of the file size. | n/a |