AoU Integration Guide

Table of Contents

AoU Solution **v2.1.11**, *2024-10-29*

The current documentation is available in HTML or PDF.

Chapter 1. AoU Architecture

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.

Chapter 2. Integration Points

2.1. For Submitting Publications

See the details in Integration for Submitting Publications section.

2.2. For Searching Publications

See the details in Integration for Searching Publications section.

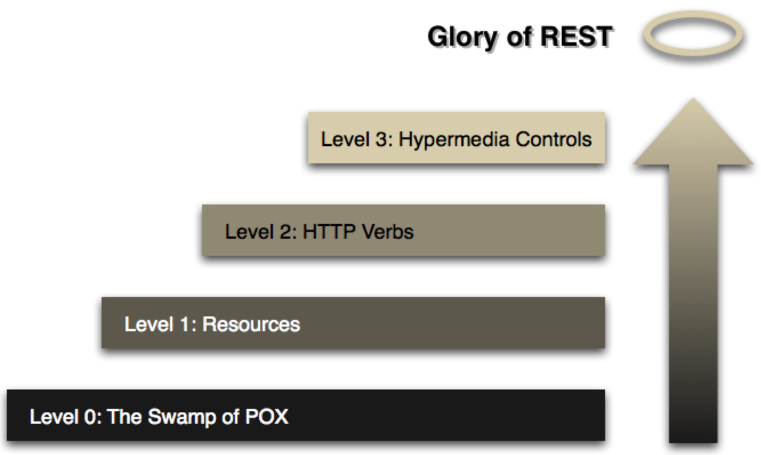
2.3. For Developers

- All web services are detailed in API Documentation.
- The API are available in OpenApi v3 format. The definition is available in AoU API JSON file.

Chapter 3. REST Web Services

3.1. Overview

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

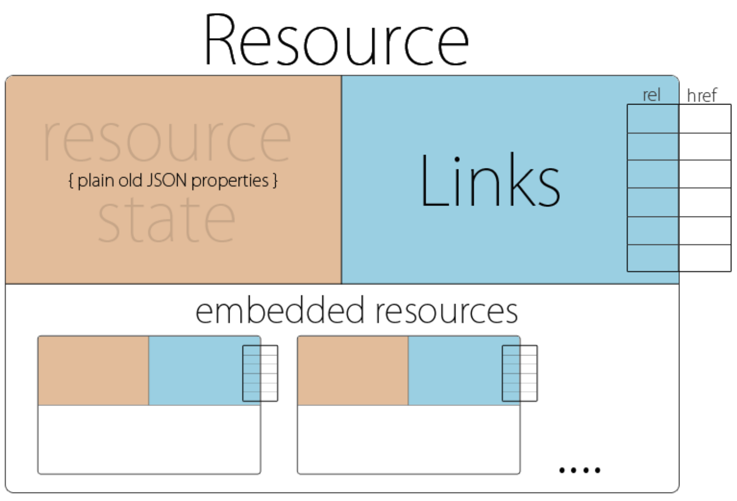


Source : (crummy.com, 2008)

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 AoU Architecture): the different module names are detailed in the AoU 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 *Used to create a new resource*	▯	▯
`GET`	Read *Used to retrieve a resource or resource list*	▯	▯
`PATCH`	Update *No creation* *Used to update an existing resource, including partial updates*	▯	▯
`DELETE`	Delete *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

Status code	Usage
`400 Bad Request`	The request was malformed. The response body will include an error providing further information
`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 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

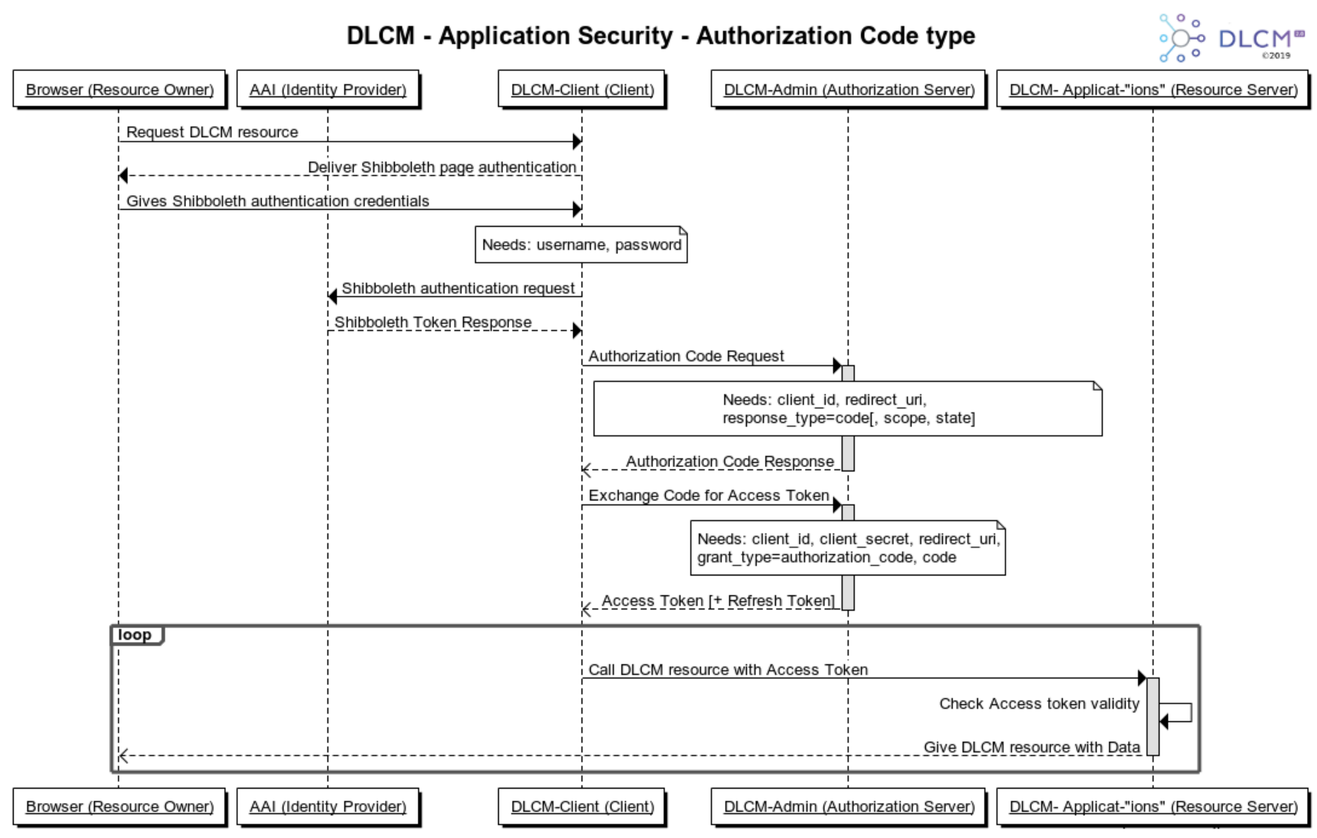
3.4.1. Authentication

All web services are secured and require authentication.

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

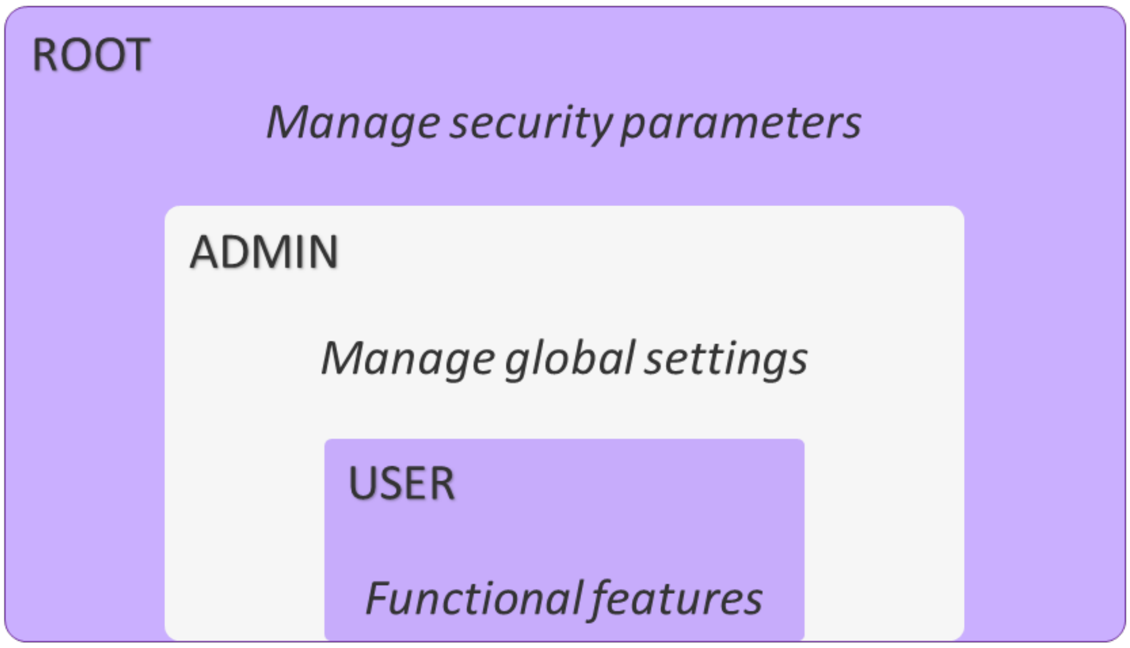
Access to Web services relies on OAuth 2.0 access delegation.

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.


DLCM - Application Security - Authorization Code type
DLCM ©2019
Browser (Resource Owner)
AAI (Identity Provider)
DLCM-Client (Client)
DLCM-Admin (Authorization Server)
DLCM- Applicat-"ions" (Resource Server)
Request DLCM resource
Deliver Shibboleth page authentication
Gives Shibboleth authentication credentials
Needs: username, password
Shibboleth authentication request
Shibboleth Token Response
Authorization Code Request
Needs: client_id, redirect_uri, response_type=code[, scope, state]
Authorization Code Response
Exchange Code for Access Token
Needs: client_id, client_secret, redirect_uri, grant_type=authorization_code, code
Access Token [+ Refresh Token]
loop
Call DLCM resource with Access Token
Check Access token validity
Give DLCM resource with Data
Browser (Resource Owner)
AAI (Identity Provider)
DLCM-Client (Client)
DLCM-Admin (Authorization Server)
DLCM- Applicat-"ions" (Resource Server)

3.4.2. Application Roles

Application Roles



DLCM 2.0 ©2019

- Functional features list

Chapter 4. Integration for Submitting Publications

Chapter 5. Integration for Searching Publications

5.1. 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 6. Annexes

6.1. Glossary

Acronym	Description	Source
CRUD	Create Read Update Delete	Software
HAL	Hypertext Application Language	Software
HATEOAS	Hypermedia As The Engine Of Application State	Software
JSON	JavaScript Object Notation	Software
REST	REpresentational State Transfer	Software
SOA	Service Oriented Architecture	Software

6.2. AoU Modules

Module	Description	REST Name
Access	*Access* module to access to publication	access
Admin	*Administration* module to manage general settings	admin