



Europeana Space – Spaces of possibility for the creative reuse of Europeana's content

CIP Best practice network - project number 621037

| | |
|---------------------------|-------------|
| Deliverable number | D2.4 |
| Title | Access APIs |

| | |
|--------------------------------------|---------------|
| Due date | Month 20 |
| Actual date of delivery to EC | 19 April 2016 |

| | | | | | | |
|---|-------------------|-------------------------------------|----------|--------------------------|-------------------|-------------------------------------|
| Included (indicate as appropriate) | Executive Summary | <input checked="" type="checkbox"/> | Abstract | <input type="checkbox"/> | Table of Contents | <input checked="" type="checkbox"/> |
|---|-------------------|-------------------------------------|----------|--------------------------|-------------------|-------------------------------------|

Project Coordinator:

Coventry University

Professor Sarah Whatley

Priority Street, Coventry CV1 5FB, UK

+44 (0) 797 4984304

E-mail: S.Whatley@coventry.ac.uk

Project WEB site address: <http://www.europeana-space.eu>

Context:

| | |
|--|---|
| Partner responsible for deliverable | NTUA |
| Deliverable author(s) | Nasos Drosopoulos (NTUA) and Frederik Temmermans (iMinds) |
| Deliverable version number | 1.4 |

| | |
|----------------------------|-------------------------------------|
| Dissemination Level | |
| Public | <input checked="" type="checkbox"/> |

History:

| Change log | | | |
|------------|------------|--|--|
| Version | Date | Author | Reason for change |
| 0.1 | 23-09-2015 | Nasos Drosopoulos | Outline & ToC |
| 0.2 | 25-09-2015 | Nasos Drosopoulos, Frederik Temmermans | First draft |
| 0.2 | 02-10-2015 | Michael Giazitzoglou | Full list of API calls and documentation |
| 0.5 | 21-11-2015 | Nasos Drosopoulos, Eirini Kaldeli, Giorgos Marinellis, Maria Ralli | Update API calls responses using feedback from hands-on session with WP4 pilot teams |
| 1.0 | 21-12-2015 | Nasos Drosopoulos | Update to correspond to the new data model for the Technical Space |
| 1.1 | 26-02-2016 | Michael Giazitzoglou | Updated list of API (v.2) calls |
| 1.2 | 30-03-2016 | Giorgos Marinellis | Updated application data model |
| 1.3 | 04-04-2016 | Nasos Drosopoulos, Frederik Temmermans | Final draft |
| 1.4 | 19-04-2016 | Nasos Drosopoulos and Tim Hammerton | Conclusion added and final amendments |

| Release approval | | | |
|------------------|------------|-----------------------|-----------------|
| Version | Date | Name & organisation | Role |
| 1.4 | 19-04-2016 | Tim Hammerton, COVUNI | Project Manager |

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

TABLE OF CONTENTS

| | |
|--|-----------|
| 1 EXECUTIVE SUMMARY | 5 |
| 2 INTRODUCTION | 6 |
| 2.1 EUROPEANA SPACE INFRASTRUCTURE CONCEPT AND COMPONENTS..... | 6 |
| 2.2 STRUCTURE OF THE DOCUMENT | 7 |
| 3 CULTURAL REPOSITORIES API COMBINATION | 8 |
| 4 TECHNICAL SPACE ACCESS APIS | 9 |
| 4.1 REQUIREMENTS, IMPLEMENTATION DETAILS AND API OVERVIEW | 9 |
| 4.2 APPLICATION DATA MODEL..... | 11 |
| 4.2.1 Resource modelling | 11 |
| 4.2.2 Main classes and properties..... | 12 |
| 4.2.3 Implementation | 18 |
| 4.3 API CALLS..... | 20 |
| 4.3.1 Search..... | 20 |
| 4.3.2 Record | 25 |
| 4.3.3 Collection and Exhibition | 34 |
| 4.3.4 User | 48 |
| 4.3.5 User Groups..... | 56 |
| 4.3.6 Organization and Project | 62 |
| 4.3.7 Media | 65 |
| 4.4 INTERACTIVE DOCUMENTATION USING SWAGGER | 67 |
| 4.4.1 API Keys | 67 |
| 4.4.2 Swagger UI | 67 |
| 5 INTEROPERABILITY WITH THE JPSEARCH FRAMEWORK..... | 73 |
| 5.1 INTRODUCTION | 73 |
| 5.2 ACHIEVING INTEROPERABILITY WITH THE JPSEARCH FRAMEWORK | 75 |
| 5.2.1 Embedding metadata in JPEG images..... | 75 |
| 5.2.2 Metadata mapping and registration..... | 75 |
| 5.2.3 Linked data..... | 76 |
| 5.3 JPSEARCH API..... | 76 |
| 5.3.1 Basic concepts | 76 |
| 5.3.2 Image resources | 77 |
| 5.3.3 Image metadata..... | 78 |
| 5.3.4 Collections | 78 |
| 6 CONCLUSIONS | 79 |

1 EXECUTIVE SUMMARY

The Europeana Space project aims to increase and enhance the creative industries' use of Europeana and other online collections of digital cultural content, by delivering a range of resources to support their engagement. The project addresses all sectors of the creative industries, from content providers to producers, exhibitors, artists and makers of cultural/creative content, publishers, broadcasters, telecoms and distributors of digital content.

The Technical Space has been made available for cultural institutions and organizations, professional users and third party developers in order to easily search for the cultural resources that meet their retrieval criteria so as to collect, use and re-use them to promote innovation and demonstrate the social and economic value of cultural content (see D2.3 – *Europeana Space Infrastructure*). This is achieved through the delivery of APIs that facilitate the development of applications based on cultural content and is validated through the realisation of the six Pilot projects. The latter serve as the basis for continued experimentation and innovation in a series of dedicated hackathons and workshops that are expected to produce new applications and services based on Europeana's resources. The interaction with this Innovation Space produces additional requirements that are evaluated and addressed by the Technical Space, reflecting a 'real-world' approach to development that can be made immediately useful.

This report documents the platform's developed APIs and serves as a user manual for professional users and third party developers who intend to use them in order to consume cultural resources for the development of applications. It presents and discusses the platform's application data model and provides an overview and detailed documentation of all available API calls. Finally, it documents guidelines and actions taken to achieve interoperability of cultural heritage image repositories with the JPSearch framework, while describing an open source implementation of the JPSearch API that is provided as a reference.

2 INTRODUCTION

2.1 EUROPEANA SPACE INFRASTRUCTURE CONCEPT AND COMPONENTS

Europeana Space introduced the Technical Space (documented in D2.3 – *Europeana Space infrastructure*) in its effort to support and promote the re-use of digital cultural heritage resources. The Technical Space is a platform for storing, accessing and processing content and metadata. It is designed and developed in alignment with complementary services used and produced in the Europeana ecosystem, and informed by the design of respective infrastructures being developed, such as Europeana Labs¹, the Europeana Cloud² and LoCloud³, and of more specialized applications targeting cultural heritage content visualization and re-use, such as the tools and pilots of AthenaPlus⁴ and EUscreen⁵. The platform targets cultural institutions and organizations, professional users and third party developers, offering the ability to easily search for the cultural resources that meet their retrieval criteria so as to collect, use and re-use them to promote innovation and demonstrate the social and economic value of cultural content.

At a high level, its functionalities can be summarized in the following list:

- Aggregate multiple sources of cultural heritage content.
- Create and curate collections of digital resources.
- Upload and add metadata and content to the search base.
- Maintain interoperability with data models and standards using the services of the metadata processing unit (MPU, see D2.2 for specifications and documentation).
- Store metadata in several formats and serialisations; support for widely used domain models.
- Serve collections as specific backends for specialized front-end applications.

The latter is achieved through the delivery of APIs that facilitate the development of applications based on cultural content and is validated through the realisation of the project's six Pilot projects. Those serve as the basis for continued experimentation and innovation in a series of dedicated hackathons and workshops that are expected to produce new applications and services based on Europeana's resources. The platform can be accessed through its landing page at <http://with.image.ntua.gr/> or via the customized group pages (see D2.3 for more details) such as the project's space⁶ or specific event pages e.g. the Publishing pilot hackathon⁷.

1 <http://labs.europeana.eu/>

2 <http://pro.europeana.eu/project/europeana-cloud>

3 <http://locloud.eu>

4 <http://www.athenaplus.eu/>

5 <http://blog.euscreen.eu/euscreenxl>

6 <http://with.image.ntua.gr/custom/espace/>

7 <http://with.image.ntua.gr/custom/hackthebook/>

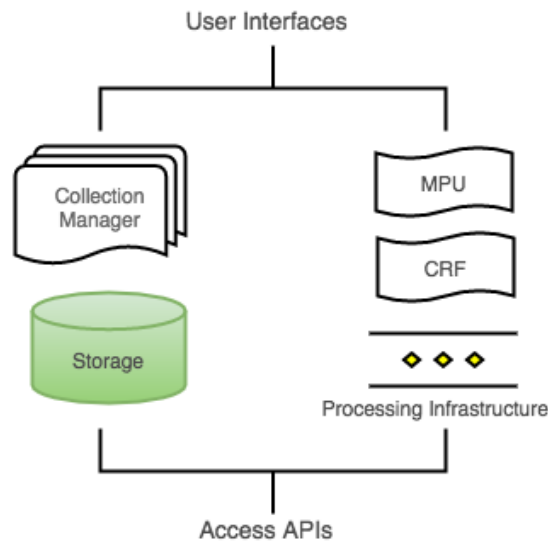


Figure 1. Europeana Space Infrastructure software components

The platform consists of the following components, illustrated in the diagram of Figure 1, and is documented in detail in deliverable D2.3 - *The Europeana Space Infrastructure*:

- The storage layer for metadata and content.
- The workflow engine for managing and publishing collections of digital resources.
- The Content Reuse Framework and the Metadata Processing Unit.
- The Processing Infrastructure and the services deployment and integration layer.
- The User Interface to access functionalities and visualize data.
- The Access APIs for the platform's data and services.

2.2 STRUCTURE OF THE DOCUMENT

Chapter 3 focuses on an important feature of the platform that empowers its API functionality, the ability to search and collect resources from external APIs and the subsequent availability of those resources through its API in an aggregated way. Chapter 4 presents the Technical Space Access APIs, starting from an overview of requirements, implementation details and categories of API calls. Section 4.2 presents and discusses the Technical Space application data model that governs the schemas and serializations for the responses of the API, while Section 4.3 documents in detail all available calls. Section 4.4 provides information regarding API Keys and illustrates the API's interactive documentation service. Chapter 5 presents guidelines and actions taken to achieve interoperability with the JPSearch framework⁸ for image search and retrieval. Finally, Chapter 6 summarizes the report and the WP's ongoing and expected activities related to the development and deployment of the Technical Space Access APIs.

8 <https://jpeg.org/jpsearch/index.html>

3 CULTURAL REPOSITORIES API COMBINATION

An important aspect of the platform's functionality and its approach towards content sourcing is the combination of third-party APIs in order to access several cultural heritage repositories through a unified interface (and subsequently data model), and to aggregate their responses. The need to consume information from many data sources is ever present today in web development and in the cultural domain specifically, with the proliferation of data sources hosting important resources in the form of metadata records, thesauri, authority files, linked data etc. The Technical Space implements a federated search that can configure, integrate and access available sources through their exposed APIs. The different results of the common search are processed and aggregated by the system before being presented to the user. This processing corresponds mainly to two categories, translation and analysis. The former includes mapping, formatting and normalization of the API responses so that they can be listed in an aggregation format, compared and re-used in a common way. The latter corresponds to comparing the different responses and information and allowing for common handling of important data such as the rights statements and reuse information. This then leads to an availability of collections of diverse resources through a single API that allows common access and visualization, and facilitates application development.

The federated search of the Technical Space currently integrates the following cultural heritage data sources:

- Europeana
- Digital Public Library of America
- National Library of Australia
- Digital New Zealand
- Rijksmuseum
- British Library collections on Flickr Commons
- Europeana Fashion
- YouTube

Most of them expose metadata records and content files for cultural heritage items with a different range in terms of geographical coverage (from continental aggregators to individual institution repositories) and types of content (Image, Text, Audio, Video etc.), while others offer more specialized resources such as people, places and thesaurus concepts, or focus on exposing higher quality content or media in general. The collections created through the federated search allow the user to combine the potentials of the different APIs to provide a single powerful new service that gives easy, homogenized access to heterogeneous information. For more details on implementation and the configuration of sources see Section 3.3.2 in D2.3.

4 TECHNICAL SPACE ACCESS APIS

4.1 REQUIREMENTS, IMPLEMENTATION DETAILS AND API OVERVIEW

The necessity for an elaborate, fully functional API for the Europeana Space infrastructure was addressed from the proposal phase of the project and subsequently highlighted in various occasions until the conclusion of the requirements analysis that is reported in deliverable 2.1. The latter also highlighted the importance of the APIs combination outlined in the previous chapter, as a first step for sourcing diverse content that can then be accessed through a common API.

The Technical Space is using NTUA's WITH, an elaborate platform for both users and developers, which was built with the Play Framework⁹, AKKA toolkit¹⁰, MongoDB database¹¹, and the Elasticsearch engine¹², using JAVA and Javascript. WITH was designed to use a REST (Representational State Transfer) API to access the back-end and communicate with the front-end, providing a way for developers of other applications to use the data and services that it exposes. This decision was supported by the requirements analysis and has added to the versatility of the platform as first usage cases, from the project's pilots and hackathon participants, have even shown the potential need of using more calls than the ones that provide access to cultural heritage resources, such as calls regarding users and groups, or to services such as the federated search. Of course, some of the more platform specific calls introduce the necessity for the API consumer to understand some of its business logic, but careful design prevents inconsistency and maintenance overheads, while the main set of calls that pertain to cultural heritage resources remain RESTful.

REST introduces a set of constraints to the design of software components that use URIs and typically communicate over HTTP. RESTful systems use a constrained vocabulary of verbs to retrieve and send data by interfacing with web resources¹³. The API is defined from its base URI, the media types used for data communication, the HTTP methods it uses, and the hypertext links for manipulating resources (e.g. create a file or update a metadata record) represented by URIs. The base URI of its API is <http://with.image.ntua.gr/> and the media type is JSON. It uses the common HTTP methods GET, PUT, POST and DELETE which operate on resources, using links like {baseURI}/collection/{collectionId}/list. Along with these requests a number of parameters can be passed to facilitate data exchange. All calls, schemas and responses have been documented in detail and are provided in the respective developers pages (accessible through the platform's UI), using the Swagger schema¹⁴ for two visualizations (see Section 4.5).

9 <https://www.playframework.com/>

10 <http://akka.io/>

11 <https://www.mongodb.org/>

12 <https://www.elastic.co/products/elasticsearch>

13 http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

14 <http://swagger.io/specification/>

Following is an overview list of the main categories of calls of the Technical Space API:

- *Search*

It includes calls for retrieving internal and external resources and additional ones that help the discoverability of available sources and filtering options. There are two search calls, a general search that only returns an array of records from external sources, and an advanced that also returns a list of aggregated filters to facilitate faceted search on both internal and external records.

- *Records and Rights*

This category refers to calls for actions on internal records, i.e. resources within the Technical Space database according to its license and the user's rights (for more information on authentication, user and IP rights see D2.3 and specifically sections 3.2.1, 3.3.1 and 4.1). It includes calls to retrieve a record's metadata, update a record or delete it. The record is specified by a parameter in the call path, the record's ID value. External search results are not kept in any database, however any records added to collections are stored.

- *Collections and Exhibitions*

Calls to collections are the main way users can search the internal records of the WITH database. Since user access to collections is controlled by their owners through access rights, it is a consequence that any search performed adheres to them. Two calls are provided that list all collections available to a logged in user. List collections shows all the collections the user has at least read access to, and list shared collections shows all collections that have been explicitly shared with the user, i.e. not public collections. These two calls can have many different filter options passed as parameters in the request. Other calls concerning collections offer basic functionalities such as create, edit or remove a collection, add or remove a record to a collection, list all the records in a collection and list all the collections of a specified user. Collections and exhibitions are treated in a similar fashion in the database, therefore calls to collections can refer to exhibitions as well.

- *Users and user groups*

Calls to manage and authenticate users and groups. Users refer to individual user accounts that have registered in WITH. User groups on the other hand can refer to either groups, organizations or projects. The calls to organizations or projects are, in essence, the same as the calls to groups, however different paths have been provided in the API for simplicity. On the other hand, some calls that refer to common elements of users and groups, like the profile id for example, are the same, so the same path is used.

- *Media*

Calls specific to managing media resources (e.g. image files) stored in the platform to enable the project's content repository. They provide functionality for creation, access, editing of metadata and deletion of media. Calls to WITH's Media Checker library¹⁵ will be included here when available (see section 3.2.2 in D2.3).

15 <https://github.com/finikm/mediachecker>

- *Other*

Other API calls exist mainly to provide specific functionalities for the platform's front-end. Those are not listed in the 'public' calls in the API documentation, even though they are available and are also being tested by specific API consumers. Such calls refer to auto-complete functions, mapping static resources like files to the URL path etc. New calls are being added as development continues, especially through the interaction with users and API consumers, and subsets that are being beta-tested, such as the semantic relation calls used by the Europeana TV pilot of the project, will eventually become available.

Finally, it is important to reiterate that most of these calls are used by the Technical Space's front-end, therefore the inspector console (especially the network tab) of all major modern browsers is a great source of examples of how these calls are being used.

4.2 APPLICATION DATA MODEL

This section presents and discusses the way WITH stores its data and implements the Technical Space application data model. This structure acts as the internal and external API, by defining the schemas for data exchange, granting developers access to the data store and exceptional control over the system's services. This document does not describe the components that access, create or consume this data to provide the functionality of the Technical Space; for more on that, see deliverable 2.3 - *Europeana Space Infrastructure*.

One of the most evolving parts of the platform is the data model for representing collected items from external repositories. The architectural choices for the platform specifically allowed for an evolving data model, as the modelling activities in the cultural heritage domain are numerous, with new schemas frequently introduced, existing standards updated, thesauri and authority files evolving and, the level and expressivity of their semantics being a parameter that is still evaluated and fine-tuned. The establishment of interoperability between the Data Infrastructure and the external sources, through the implementation of formal mappings between them, allows for updating the data model and extracting any additional information or updating the crosswalk to correspond to the improved alignment. In that way the platform's aggregation data model can be developed further without interrupting or disrupting the platform's usage.

During the second year of the project a new data model for the platform was implemented, which allows a big step to be taken towards a more powerful and expressive system. There are several updates that focus on enabling better ways of combining sources and applying common filters and criteria when searching for cultural heritage material in the evolving set of external sources.

4.2.1 Resource modelling

As reported in D2.3, it was decided to redesign the original data model of the platform, based on the decision to properly model the collection of web resources, instead of what is traditionally termed as 'items'. Currently, Europeana, DPLA and such aggregation repositories, collect and expose metadata records about cultural heritage objects or items. In those records CH objects are connected to other resources, such as the media files that depict them and the Agents (people or organizations), Places, Timespans or Concepts that appear in their documentation.

There are also several, very important sources that only expose such resources not in connection with an item but standalone, for example thesauri of concepts, datasets describing famous artists, geographical databases and so on. The new data model allows for searching and collecting all kinds of resources instead of only CH objects, giving the ability to create collections that are not just sets of records, but a group of web resources that are semantically linked. In that sense, a user that is interested, for example, in Italian Renaissance painting, can collect famous paintings (items from CH aggregators) together with resources that represent the painters themselves (e.g. DBpedia topics or VIAF authority files), the locations where they were working or exhibiting (e.g. GeoNames entries), a relevant time period (<http://dbpedia.org/page/Category:Renaissance>), a set of concepts that describe the period or movement and so on.

This more expressive representation can provide an advantage to the applications that consume the data. Moreover, the semantic store and engine of the Data Infrastructure can expose more expressive structures and serializations, and perform reasoning tasks, serving as a basis for development of applications that aim at being part of the semantic web and/or linked data cloud. It also strengthens the impact of the Technical Space as a platform for re-organizing, re-purposing and re-using cultural heritage by moving away from the notion of item repositories towards a network of connected resources. In the same sense the data model also implements a structure for recording events (combinations of objects, agents, places and timespans, as defined by CIDOC-CRM and used by the LIDO schema) in order to properly represent and visualize information about cultural heritage resources. Since established standards for modelling cultural heritage resources exist, for example EDM and LIDO, the application data model reuses these vocabularies, while the API responses can be serialized using these formats.

4.2.2 Main classes and properties

The main data structure in WITH's data model is the `WithResource`. Everything collected, or produced through MINT and pushed to WITH is a `WithResource` that can be discovered, viewed, collected and re-used. At the moment the following classes extend the `WithResource` class:

- `Collection`
- `CulturalObject`
- `Place`
- `Event`
- `Agent`
- `Timespan`

Every `WithResource` is described using the following metadata categories that are defined in more detail in the next section:

- `Administrative`
- `Usage`
- `Provenance`

- Collection related
- Media
- Descriptive

Administrative (all the elements are mandatory)

| Element | Type | Description/Comments |
|-----------------|-----------------------|---|
| access | WithAccess | User ID and types/level of access |
| withCreator | ObjectId | The identifier of the creator of the resource in WITH |
| withURI | URI | Generated URI for the resource in WITH domain |
| created | Date | The date the resource created in WITH |
| lastModified | Date | The date the resource was last modified in WITH |
| underModeration | Map<ObjectId, Access> | Reserved for future implementation of crowdsourced moderated editing or enrichment of resources |
| externalId | String | This identifier is used for resolving duplicates. It is a hash of the source, provider and the external ID created from the last entry in the provenance chain. |

Usage (social networking dimensions)

| Element | Type | Description/Comments |
|-----------|-------------------|--|
| likes | Integer | The number of times the collected resource got a like. |
| collected | Integer | The number of times a resource is collected. |
| annotated | Integer | The number of times a resource is annotated. |
| viewCount | Integer | The number of times the resource was viewed |
| tags | ArrayList<String> | User or automatically assigned tags with no semantics |

Provenance

| Element | Type | Description/Comments |
|------------|---|---|
| datasource | String one of: Europeana, DPLA, DigitalNZ etc. | The datasource from which the resource was collected. The range of the element is an enumeration of the datasources supported. |

| | | |
|--------------------|------------------------------------|--|
| source | String XML or JSON | The original record and serialization of the collected resource. |
| datasourceRecordID | String | The identifier of the resource in the data source from which it was collected by the WITH user e.g. the Europeana identifier |
| provider | String, source dependent | The provider of the resource. This varies for each data source. In the case of Europeana and DPLA this can be the Aggregator. |
| dataproducer | String | The data provider of the resource. This is usually the organization that created and hosts the collected resource. |
| originalId | Array<String> | The original identifier of the resource. In most cases the identifier is one, but there are quite a few cases that more than one local identifiers are used (values of dc:identifier). |
| originalURL | String URL, Source dependent | The original URL of the resource. In the case of the Europeana source this will be the isShownAt value. |

Collection Related

| Element | Type | Description/Comments |
|--------------|--------|--|
| collectionID | String | Normally the range of this element should be a URI. The value for WITH collections is a hash ID). In addition, a collected resource may already belong to an external collection that can be represented here. |
| position | String | The position of the collectedResource in the collection. |

Media objects are created for each digital object linked to a resource

| Element | Type |
|----------------|-------------------|
| dbId | ObjectId |
| resources | Array<ObjectId> |
| MimeType | Media type |
| OriginalRights | LiteralorResource |
| Width | Int |

| | |
|--------------------|----------|
| Height | Int |
| Size | Long |
| mediaBytes | byte |
| Codec | String |
| durationSeconds | Double |
| SpatialResolution | Int |
| BitRate | Int |
| FrameRate | Int |
| ColorSpace | String |
| ComponentColor | Hex |
| AudioChannelNumber | Int |
| SampleSize | Int |
| SampleRate | Int |
| ParentID | ObjectId |

Descriptive

Descriptive metadata vary depending on the source and type of the **collected resource**. The properties in the next table constitute the minimum information required for a resource in WITH. Semantics for elements are inherited by the vocabularies reused. Mandatory elements:

- resourceType
- one of title, description
- one of edm:isShownBy, isShownAt
- MetadataRights

| Element / Semantics | Type | Description/Comments |
|--|------------|---|
| Label <i>dc:title or skos:prefLabel</i> | Literal(s) | The label of the resource. This is the title in case of objects, and name in case of Agents and Places. |
| Description <i>dc:description</i> | Literal(s) | A description of the resource. |
| Alternative <i>dct:alternative or</i> | Literal(s) | The alternative label of the resource. |

| | | |
|------------------------------------|--------------------------|--|
| <i>skos:altLabel</i> | | |
| Keywords <i>dc:subject</i> | Array<ResourceOrLiteral> | The subject of the resource and possible tags. |
| IsShownBy see EDM | URL | Digital object |
| IsShownAt see EDM | URL | HTML representation |
| MetadataRights <i>dc:rights</i> | ResourceOrLiteral | Rights statements, licenses |
| sameAs | Array<String/URIs> | URL of a reference Web page that unambiguously indicates the item's identity. E.g. the URL of the item's Wikipedia page, Freebase page, or official website. |
| rdfType | Array<String/URIs> | An element for additional RDF/OWL classifications. |
| MetadataQuality | | Quality metrics |

PCHO stands for **Provided Cultural Heritage Object** and corresponds to the majority of the resources collected through WITH. PCHO extends the collected resource with the elements of the following table.

| Element | Type | Description/Comments |
|--------------|--------------------------|---|
| dcidentifier | Array<ResourceOrLiteral> | The local identifier. |
| dclanguage | Array<Literal> | The language. |
| dctype | Array<ResourceOrLiteral> | The nature or genre of the resource. Type includes terms describing general categories, functions, genres, or aggregation levels for content. |
| dccoverage | Array<ResourceOrLiteral> | The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant. This may be a named place, a location, a spatial coordinate, a period, date, date range or a named administrative entity. |
| dctspatial | Array<ResourceOrLiteral> | Spatial characteristics of the resource. |
| dccreator | Array<ResourceOrLiteral> | An entity primarily responsible for making the resource. This may be a person, organisation or a service. |

| | | |
|-------------|--------------------------|--|
| dccreated | Array<WithTime> | |
| dcdte | Array<WithTime> | A point or period of time associated with an event in the lifecycle of the resource. |
| dcformat | Array<ResourceOrLiteral> | The file format, physical medium or dimensions of the resource. |
| dctmedium | Array<ResourceOrLiteral> | The material or physical carrier of the resource. |
| isRelatedTo | Array<Resource> | The related resources of the given resource. |
| events | Array<CidocEvent> | Events |

Agent extends the collected resource with the elements of the following table.

| Element | Type | Description/Comments |
|--------------|--------------------------|---|
| resourceType | String | The type of the collected resource. In this case the value will be "Agent". |
| birthdate | Array<WithDate> | The date of birth. It is an array because the exact date of birth is not always known and in many cases estimates are used. |
| birthplace | Array<ResourceOrLiteral> | The place of birth. |
| deathdate | Array<WithDate> | The date of death. It is an array because the place of birth is not always known and estimates are used. |
| deathPlace | Array<ResourceOrLiteral> | Spatial characteristics of the resource. |
| gender | Literal | |

Place extends the collected resource with the elements of the following table.

| Element | Type | Description/Comments |
|--------------|--------------------------|---|
| resourceType | String | The type of the collected resource. In this case the value will be "Place". |
| nation | Array<ResourceOrLiteral> | The nation to which the place belongs |
| continent | Array<ResourceOrLiteral> | The continent to which the place belongs. |
| partOfPlace | Array<ResourceOrLiteral> | The list of places to which the place belongs. |
| wgs84poslat | Double | The latitude coordinate. |

| | | |
|--------------|--------|---------------------------|
| wgs84poslong | Double | The longitude coordinate. |
| wgs84posalt | Double | The altitude coordinate. |
| accuracy | Double | |

TimeSpan extends the collected resource with the elements of the following table.

| Element | Type | Description/Comments |
|--------------|-------------------|--|
| resourceType | String | The type of the collected resource. In this case the value will be "TimeSpan". |
| periods | Array<WithPeriod> | The exact start date may not be known that's why an array is used. |

Event extends the collected resource with the elements of the following table.

| Element | Type | Description/Comments |
|-----------------|--------------------------|---|
| resourceType | String | The type of the collected resource. In this case the value will be "Event". |
| period | Array<WithPeriod> | The exact start date may not be known that's why an array is used. |
| personsInvolved | Array<ResourceorLiteral> | The persons involved in the event. Agents or Literals. |
| places | Array<ResourceorLiteral> | Places involved in the event. Places or Literals. |
| objectsInvolved | Array<ResourceorLiteral> | The objects involved in the event. |

4.2.3 Implementation

The diagrams of Figures 2 and 3 illustrate the current implementation of the data model in WITH's storage layer. Naturally, it goes beyond the basic specification provided in the previous sections, both due to the model and platform evolving in the same time and often independently and, due to the fact that the platform usually needs to store more, application-specific information. In Section 4.3 that documents the API calls, one can find the actual structures that an API consumer receives.

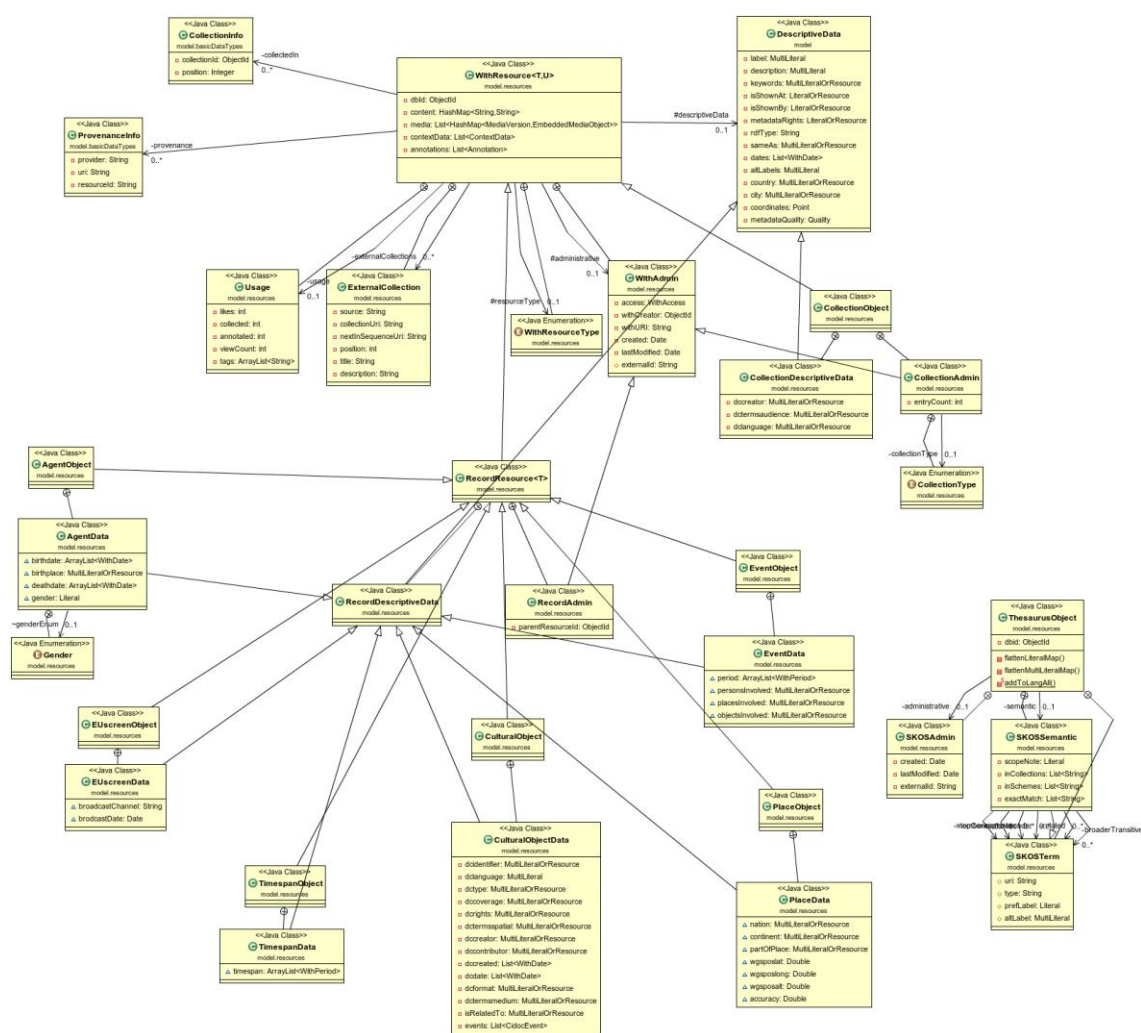


Figure 2. Main data model classes, properties and enumerations implemented for the Technical Space

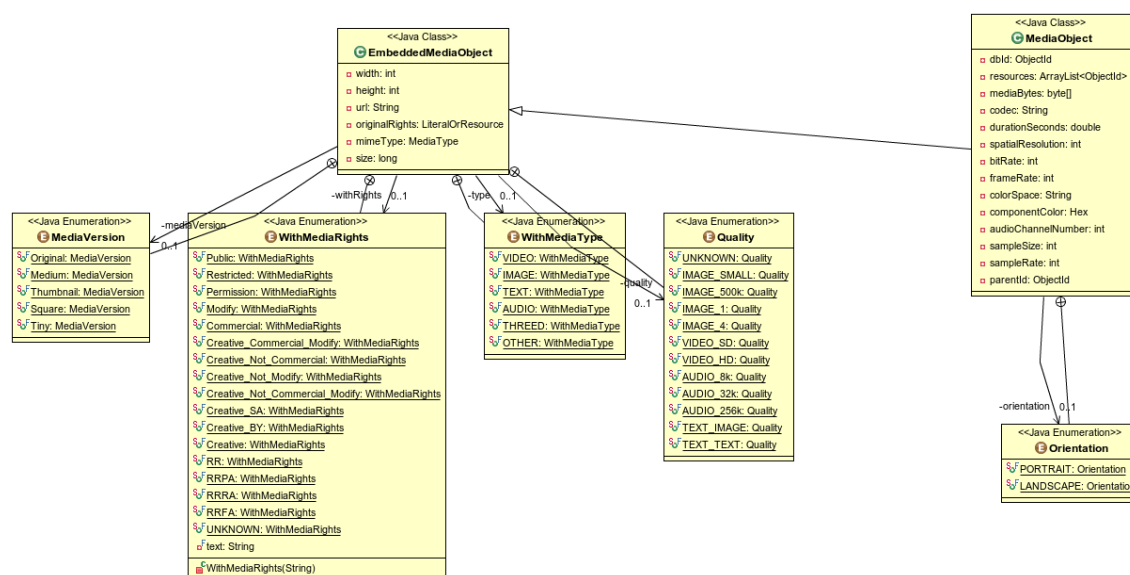


Figure 3. Media object classes, properties and enumerations

4.3 API CALLS

4.3.1 Search

General search in external resources and the WITH database

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|-------------|--|
| POST | /api/search | Body contains search parameters, response is a JSON array of records that match the search term. Boolean search supports use of AND, OR and NOT operators. Terms separated without an operator (using a space) are treated as an AND. Use of quotes will perform exact term or phrase searches. For example, "Olympian Zeus" will search for the exact phrase, whereas Olympian Zeus will equate to Olympian AND Zeus. For more search options use advanced search. |
| <i>Body Sample</i> | | <i>Response Sample</i> |
| <pre>{ "searchTerm": "zeus", "page": 1, "pageSize": 20, "source": ["Europeana"] }</pre> | | <pre>[{ "query": "string", "totalCount": 0, "startIndex": 0, "count": 0, "items": [{}], "source": "string", "facets": [{ "name": "string", "fields": [{ "label": "string", "count": 0 }] }], "filters": [{ "filterName": "string", "filterID": "string", "suggestedValues": [{}] }] }]</pre> |

| | |
|--|--|
| | <pre> "value": "string", "count": 0 }] }]] } }] </pre> |
| | Response Messages |
| | 400 Bad request (invalid json) |

Advanced search with filters (faceted search)

| Method | Call | Description |
|---|---------------------|--|
| POST | /api/advancedsearch | An extension of the simple search to include filters for search with facets. The values of the filters are NOT preset. To see what filters you can use, you should use the /api/initialfeatures call. Then you can send another query with some values of all the possible filters. To get a better understanding a suggestion is to use the inspector console of your favourite browser and observe the network activity while performing a search from the front-end, using different filter values. |
| Body Sample | | Response Sample |
| <pre> { "searchTerm": "zeus", "page": 1, "pageSize": 10, "source": ["Europeana"], "filters": [{ "filterID": "media.type", "values": ["image"] }] } </pre> | | <pre> [{ "responses": [{ "query": "string", "totalCount": 0, "startIndex": 0, "count": 0, "items": [{}], "source": "string", "facets": [{ "name": "string", "fields": [{ "label": "string", </pre> |

| | | |
|--|---|----------------------------|
| | <pre> "count": 0 }] }, "filters": [{ "filterName": "string", "filterID": "string", "suggestedValues": [{ "value": "string", "count": 0 }] }] }, "filters": [{ "filterName": "string", "filterID": "string", "suggestedValues": [{ "value": "string", "count": 0 }] }] }] </pre> | |
| | Response Messages | |
| | 400 | Bad request (invalid json) |

Retrieve initial filters for a search

| Method | Call | Description |
|--------|---------------------|--|
| POST | /api/initialfilters | Retrieve a JSON with the filters you can use for faceted search in the query. This call requires a body parameter with the sources you will need to process. |

| <i>Body Sample</i> | <i>Response Sample</i> |
|--|---|
| <pre>{ "source": ["Europeana"] }</pre> | <pre>[[{ "filterName": "string", "filterID": "string", "suggestedValues": [{ "value": "string", "count": 0 }] }]]</pre> |
| | Response Messages |
| | 400 Bad request (invalid json) |

Retrieve the list of available search sources

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|-------------------|--------------------|---|
| GET | /api/searchsources | Retrieve a JSON with the list of sources that WITH currently supports for federated searches. |
| Parameters | | Response Sample |
| | | <pre>["string"]</pre> |
| | | Response Messages |
| | | 400 Bad request (invalid json) |

DBPedia Lookup

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|------------------------|--|
| GET | /source/dbpedia/lookup | Returns an array of dbpedia entities that match a given term and type query. Each returned result contains the terms of the query in its label field, and belongs to at least one of the specified types. The type is a dbpedia ontology class, without the dbpedia prefix (e.g. Person, Place, etc). |
| <i>Parameters</i> | | <i>Response Sample</i> |
| query (required): string the desired query type (required): string a comma separated list of the types of the result. start : integer offset (default 0) count : integer count (default 10) | | /source/dbpedia/lookup?type=Person&query=da Vinci&start=0&count=5 <pre>{ "totalcount": 5, "results": [{ "uri": "http://dbpedia.org/resource/Paul_Da_Vinci", "label": { "en": "Paul Da Vinci" }, "abstract": { "en": "Paul Da Vinci (born Paul Leonard Prewer, 1951) is a British singer and musician. He is best known as the lead singer on the 1974 hit recording by The Rubettes, \"Sugar Baby Love\", although he did not perform with the group at the time. He reportedly had a three-and-a-half octave vocal range, and worked as a demo and session singer before and after his own moderately successful solo career, which included the UK hit \"Your Baby Ain't Your Baby Anymore.\""} }, "birthplace": ["http://dbpedia.org/resource/England", "http://dbpedia.org/resource/Essex", "http://dbpedia.org/resource/Thurrock"], "type": ["http://dbpedia.org/ontology/Person", "http://dbpedia.org/ontology/MusicalArtist", "http://dbpedia.org/ontology/Artist", "http://dbpedia.org/ontology/Agent"], "subject": ["http://dbpedia.org/resource/Category:1951_births", "http://dbpedia.org/resource/Category:English_male_singers", "http://dbpedia.org/resource/Category:English_pop_singers", "http://dbpedia.org/resource/Category:Living_people"], ... }] }</pre> |
| | | <i>Response Messages</i> |
| | | 500 Internal Server Error (lookup failure) |

4.3.2 Record

Retrieve a record

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|--------------------|--|
| GET | /record/{recordId} | Retrieve a JSON with the metadata of the record specified in the path, by its Id. To retrieve the Ids of records in collections, you can use the collection/{colId}/list call and see the dbid field in the response. Many different serializations are available for the record's metadata. To select one, or more of them, use the "format" parameter. The different serializations will appear in the "content" field in the record response. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| recordId: string The id of the record format: array The serialization of the response. One of the following: JSON_UNKNOWN, JSONLD_UNKNOWN, XML_UNKNOWN, JSON_EDM, JSONLD_EDM, XML_EDM, JSONLD_DPLA, JSON_NLA, XML_NLA, JSON_DNZ, XML_DNZ, JSON_YOUTUBE, "UNKNOWN", "all". If not specified, no content is returned, only basic collection fields. | | <pre>{ "dbId": "string", "resourceType": "WithResource", "collectedIn": [{ "collectionId": "string", "position": 0 }], "administrative": { "withURI": "string", "access": { "isPublic": true, "acl": [{ "user": "string", "level": "READ" }] } }, "withCreator": "string", "created": "string", "lastModified": "string", "underModeration": { "key": { "isPublic": true, "acl": [{ "user": "string", "level": "READ" }] } } }</pre> |

| | |
|--|--|
| | <pre> } }, "externalId": "string", "parentResourceID": "string" }, "usage": { "likes": 0, "collected": 0, "annotated": 0, "viewCount": 0, "tags": ["string"] }, "media": [{ "type": "string", "withRights": "Public", "withUrl": "string", "url": "string", "height": 0, "width": 0, "mediaVersion": "string", "originalRights": { "uri": "value" }, "mimeType": "string", "size": 0, "quality": "UNKNOWN", "dbid": "string", "mediaBytes": "string", "codec": "string", "durationSeconds": 0, "spatialResolution": 0, "bitRate": 0, "frameRate": 0, "colorSpace": "string", "componentColor": "string", "orientation": "string", "audioChannelNumber": 0, "sampleSize": 0, "sampleRate": 0 }], "descriptiveData": { "label": { "default": ["string" </pre> |
|--|--|

| | |
|--|--|
| | <pre>], "en": ["string"] }, "description": { "default": ["string"], "en": ["string"] }, "keywords": { "default": ["string"], "en": ["string"] }, "isShownAt": { "uri": "value" }, "isShownBy": { "uri": "value" }, "metadataRights": { "uri": "value" }, "rdfType": "string", "dctype": { "default": ["string"], "en": ["string"] }, "sameAs": { "default": ["string"], "en": ["string"] }, "dates": [</pre> |
|--|--|

| | |
|--|---|
| | <pre> "string"], "altLabels": { "default": ["string"], "en": ["string"] }, "dccreator": { "default": ["string"], "en": ["string"] }, "dctermsaudience": { "default": ["string"], "en": ["string"] }, "dclanguage": { "default": ["string"], "en": ["string"] }, "dccoverage": { "default": ["string"], "en": ["string"] }, "dccrights": { "default": ["string"], "en": ["string" </pre> |
|--|---|

| | |
|--|--|
| | <pre>] }, "dctermsSpatial": { "default": ["string"], "en": ["string"] }, "dccontribution": { "default": ["string"], "en": ["string"] }, "dcformat": { "default": ["string"], "en": ["string"] }, "dctermsMedium": { "default": ["string"], "en": ["string"] }, "dclsRelatedTo": { "default": ["string"], "en": ["string"] }, "dcreated": [{ "isoDate": "string", "year": 0, "epoch": { "uri": "value" } }] } </pre> |
|--|--|

```

    },
    "approximation": 0,
    "uri": "string",
    "free": "string"
  }
],
"dcddate": [
  {
    "isoDate": "string",
    "year": 0,
    "epoch": {
      "uri": "value"
    },
    "approximation": 0,
    "uri": "string",
    "free": "string"
  }
]
},
"withCreatorInfo": {
  "username": "string",
  "favorites": "string",
  "dbId": "string",
  "organizations": [
    {
      "id": "string",
      "username": "string",
      "friendlyName": "string"
    }
  ],
  "projects": [
    {
      "id": "string",
      "username": "string",
      "friendlyName": "string"
    }
  ],
  "usergroups": [
    {
      "id": "string",
      "username": "string",
      "friendlyName": "string"
    }
  ],
  "usergroupsIds": [
    "string"
  ],
  "adminGroups": [

```

| | |
|--|---|
| | <pre> "string"] }, "provenance": [{ "provider": "string", "uri": "string", "resourceId": "string" }], "content": { "key": "value" } } </pre> |
| | Response Messages |
| | 404 Not found (format not found) 500 Internal Server Error (database error) |

Update a record

| Method | Call | Description |
|--|--------------------|--|
| PUT | /record/{recordId} | Update the metadata of an existing record, specified by its id in the path. You only need to provide the fields you want updated in the record body. |
| Parameters | | Response Sample |
| recordId: string The id of the record body: InternalRecord Content type: application/json A JSON with the updated metadata | | An object record in JSON (see previous call) |
| | | Response Messages |
| | | 400 Bad Request (invalid json) 403 Forbidden (no edit permissions) 500 Internal Server Error (database error) |

Get a list of your favourite records

| Method | Call | Description |
|--------|-----------------------|--|
| GET | /collection/favorites | This will return an array of Ids for all the favoured records of the logged in user. |

| <i>Parameters</i> | <i>Response Sample</i> |
|-------------------|---|
| | ["string"] |
| | <i>Response Messages</i> |
| 400 | Bad Request (user or group cannot be found) |

Retrieve all records in a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|---------------------------------|---|
| GET | /collection/{collectionId}/list | Retrieves all records from the collection specified in the path and returns an array of record objects. Record metadata can be available in different serializations. The format parameter defines this serialization in the records array field of the JSON response. See GET /record/{recordId}. If the format parameter is empty, only the internal record will be provided in the WITH model. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionId (required): string Id of the collection start : integer offset count : integer count (default 10) format : string One of the following: JSON_UNKNOWN, JSONLD_UNKNOWN, XML_UNKNOWN, JSON_EDM, JSONLD_EDM, XML_EDM, JSONLD_DPLA, JSON_NLA, XML_NLA, JSON_DNZ, XML_DNZ, JSON_YOUTUBE, "UNKNOWN", "all". If not specified, no content is returned, only basic collection fields. | | <pre>{ "entryCount": 0, "records": [{}] }</pre> |
| | | <i>Response Messages</i> |
| | | 403 Forbidden (invalid collection id, no read access) 500 Internal Server Error (cannot retrieve records from database) |

Create a new record and add it to a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|--------------------------------------|---|
| POST | /collection/{collectionId}/addRecord | <p>Adds a record to the collection specified in the path, creating a new internal record that contains the submitted metadata. You will need to be logged in and have write access or be the owner of the collection in order to add records to it. Note that calls to this path can also be used for exhibitions. Position is a Mandatory field for exhibitions, the default is 0 i.e. the record will be displayed first.</p> <p>IMPORTANT</p> <p>This is the preferred way to create a new record! Different serializations for the new record will be created automatically and can be retrieved with GET /record/{recordId}. You do not need to provide values for all the available fields. Finally, please note that this call will only return a message. To see the collection and the new record, you will need to use GET /collection/{collectionId}/list and parse the results for a list of the record metadata in the collection and the corresponding record dbids. For individual record metadata, use GET /record/{recordId} with the dbids retrieved.</p> |
| <i>Parameters</i> | | <i>Response Sample</i> |
| <p>collectionID (required): string Id of the collection</p> <p>position: integer offset</p> <p>body: InternalRecord Content type: application/json Record JSON schema</p> | | <pre>{ "message": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | <p>400 Bad Request (no position in exhibition, constraint violation)</p> <p>403 Forbidden (no permission to edit collection)</p> <p>500 Internal Server Error (cannot save to database)</p> |

Remove a record from a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|---|---|
| DELETE | /collection/{collectionId}/removeRecord | Removes the record in the specified position, from a specified collection (path). Note that calls to this path can also be used for exhibitions. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionID (required): string Id of the collection position : integer offset recordID (required): string Id of the record | | <pre>{ "message": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 403 Forbidden (no permission to edit collection) 500 Internal Server Error (no or wrong record Id, cannot delete from database, exception error) |

4.3.3 Collection and Exhibition

Get a list of collections

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|------------------|---|
| GET | /collection/list | This call returns a list of all the accessible collections (at least read access) to the logged in user. Using the parameter filters, you can narrow down the collections associated with a specific user. All filters are optional. Since this call has many parameters it is suggested that all descriptions are worked through ahead of use. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| directlyAccessedByUserOrGroup : string <pre>{ "username": "name", "rights": "OWN" }</pre> Filters returned collections based on the specified rights for each user in the input array of JSON objects. Each user needs to have direct access rights to this collection and not through a user group (organization or project). | | <pre>{ "collectionsOrExhibitions": [{}] }</pre> |
| | | <i>Response Messages</i> |
| recursivelyAccessedByUserOrGroup : string Works like | | 500 Internal Server Error |

| | |
|---|--|
| <p>directlyAccessedByUserName with the difference that access rights can be inherited from a user group to which a user belongs.</p> <p>creator: string Filters retrieved collections based on their "ownerId" field, i.e. the Id of the user that created the collection.</p> <p>isPublic: boolean If true returns only public collections, if false or unspecified this filter remains inactive.</p> <p>IsExhibition: boolean If true returns only exhibitions, if false only collections and if unspecified returns both.</p> <p>CollectionHits: boolean If true returns the total numbers of exhibitions and/or collections this call may return (limited by the count). Default is 'false'.</p> <p>offset: integer Offset.</p> <p>count: integer Count (default 10).</p> | |
|---|--|

Get a list of collections shared with the user

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---------------|------------------------|---|
| GET | /collection/listShared | This call returns a list of all the collections that have been shared with the logged-in user. This means, the user did not create these collections but has been given access rights to it. The filters are similar to the collection/list call. Using the parameter filters, you can narrow down the collections associated with a specific user. All filters are optional. |

| <i>Parameters</i> | <i>Response Sample</i> |
|--|---|
| <p>direct: boolean If set to true, only collections shared directly (i.e. not via user groups) with the logged-in user will be shown. Default value is false.</p> | <pre>{ "collectionsOrExhibitions": [{}] }</pre> |
| <p>directlyAccessedByUserOrGroup: string { "username": "name", "rights": "OWN" } Filters returned collections based on the specified rights for each user in the input array of JSON objects. Each user needs to have direct access rights to this collection and not through a user group (organization or project).</p> <p>recursivelyAccessedByUserOrGroup: string Works like directlyAccessedByUserName with the difference that access rights can be inherited from a user group to which a user belongs.</p> <p>creator: string Filters retrieved collections based on their "ownerId" field, i.e. the Id of the user that created the collection.</p> <p>IsExhibition: boolean If true returns only exhibitions, if false only collections and if unspecified returns both.</p> <p>CollectionHits: boolean If true returns the total numbers of exhibitions and/or collections this call may return (limited by the count). Default is 'false'.</p> <p>offset: integer Offset.</p> <p>count: integer Count (default 10).</p> | <p><i>Response Messages</i></p> <p>403 Forbidden (user not specified)</p> |

Get a list of your favourite records

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|-------------------|-----------------------|--|
| GET | /collection/favorites | This will return an array of Ids for all the favoured records of the logged in user. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| | | ["string"] |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (user or group cannot be found) |

Retrieve all users that have access to a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|--------------------------------------|--|
| GET | /collection/{collectionId}/listUsers | Retrieves an array with all the users that have access to this collection and their access rights. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionID (required): string Id of the collection. | | [[{ "category": "user", "userId": "string", "firstName": "string", "lastName": "string", "username": "string", "accessRights": "READ" }]] |
| | | <i>Response Messages</i> |
| | | 403 Forbidden (invalid collection id, no read access) 500 Internal Server Error (cannot retrieve records from database) |

Delete a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|----------------------------|---|
| DELETE | /collection/{collectionId} | Removes a collection from the database. Records that were created into this collection will also be deleted. Note that calls to this path can also be used for exhibitions. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionID (required): string Id of the collection | | <pre>{ "message": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 403 Forbidden (no read-access) 500 Internal Server Error (database error) |

Retrieve collection metadata

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|----------------------------|---|
| GET | /collection/{collectionId} | Returns the metadata of the collection specified in path. Note that calls to this path can also be used for exhibitions. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionID (required): string Id of the collection. | | <pre>{ "dbId": "string", "resourceType": "WithResource", "collectedIn": [{ "collectionId": "string", "position": 0 }], "administrative": { "withURI": "string", "access": { "isPublic": true, "acl": [{ "user": "string", "level": "READ" }] } } },</pre> |

| | |
|--|---|
| | <pre> "withCreator": "string", "created": "string", "lastModified": "string", "underModeration": { "key": { "isPublic": true, "acl": [{ "user": "string", "level": "READ" }] } }, "externalId": "string", "parentResourceID": "string" }, "usage": { "likes": 0, "collected": 0, "annotated": 0, "viewCount": 0, "tags": ["string"] }, "media": [{ "type": "string", "withRights": "Public", "withUrl": "string", "url": "string", "height": 0, "width": 0, "mediaVersion": "string", "originalRights": { "uri": "value" }, "mimeType": "string", "size": 0, "quality": "UNKNOWN", "dbid": "string", "mediaBytes": "string", "codec": "string", "durationSeconds": 0, "spatialResolution": 0, "bitRate": 0, "frameRate": 0, </pre> |
|--|---|

| | |
|--|--|
| | <pre> "colorSpace": "string", "componentColor": "string", "orientation": "string", "audioChannelNumber": 0, "sampleSize": 0, "sampleRate": 0 }], "descriptiveData": { "label": { "default": ["string"], "en": ["string"] }, "description": { "default": ["string"], "en": ["string"] }, "keywords": { "default": ["string"], "en": ["string"] }, "isShownAt": { "uri": "value" }, "isShownBy": { "uri": "value" }, "metadataRights": { "uri": "value" }, "rdfType": "string", "dctype": { "default": ["string"], "en": [</pre> |
|--|--|

| | |
|--|--|
| | <pre> "string"] }, "sameAs": { "default": ["string"], "en": ["string"] }, "dates": ["string"], "altLabels": { "default": ["string"], "en": ["string"] }, "dccreator": { "default": ["string"], "en": ["string"] }, "dctermsaudience": { "default": ["string"], "en": ["string"] }, "dclanguage": { "default": ["string"], "en": ["string"] }, "dccoverage": { "default": [</pre> |
|--|--|

| | |
|--|--|
| | <pre> "string"], "en": ["string"] }, "dcrights": { "default": ["string"], "en": ["string"] }, "dctermsSpatial": { "default": ["string"], "en": ["string"] }, "dccontribution": { "default": ["string"], "en": ["string"] }, "dcformat": { "default": ["string"], "en": ["string"] }, "dctermsMedium": { "default": ["string"], "en": ["string"] }, "dclsRelatedTo": { "default": [</pre> |
|--|--|

| | |
|--|--|
| | <pre> "string"], "en": ["string"] }, "dccreated": [{ "isoDate": "string", "year": 0, "epoch": { "uri": "value" }, "approximation": 0, "uri": "string", "free": "string" }], "dcdde": [{ "isoDate": "string", "year": 0, "epoch": { "uri": "value" }, "approximation": 0, "uri": "string", "free": "string" }] }, "withCreatorInfo": { "username": "string", "favorites": "string", "dbld": "string", "organizations": [{ "id": "string", "username": "string", "friendlyName": "string" }], "projects": [{ "id": "string", "username": "string", "friendlyName": "string" }] </pre> |
|--|--|

| | | |
|--|--|---------------------------------------|
| | <pre>], "usergroups": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroupsIds": ["string"], "adminGroups": ["string"] }, "provenance": [{ "provider": "string", "uri": "string", "resourceId": "string" }], "content": { "key": "value" } } </pre> | |
| | Response Messages | |
| | 403 | Forbidden (no read-access) |
| | 500 | Internal Server Error(database error) |

Create a new collection

| Method | Call | Description |
|---------------|-------------|---|
| POST | /collection | This call will create a new collection with an item and store it in the database. Please use with caution as testing with this call will update your current collections! |

| <i>Parameters</i> | <i>Response Sample</i> |
|---|---|
| CollectionType: string Simple collection (default) or exhibition body (required): collection record Content type: application/json | <pre>{ "collection": {} }</pre> |
| | <i>Response Messages</i> |
| | 403 Forbidden (no read-access) 500 Internal Server Error(database error) |

Update metadata in a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|----------------------------|---|
| POST | /collection/{collectionId} | Use this call to change the stored metadata of a collection. Note that calls to this path can also be used for exhibitions. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| collectionID (required): string Id of the collection body (required): collection record Content type: application/json New collection/exhibition metadata. Only provide the fields you wish to be changed. | | <pre>{ "owner": "string", "access": "string", "collection": {} }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (null/invalid JSON, duplicate title, wrong JSON fields) 403 Forbidden (no read-access) 500 Internal Server Error (database error) |

Change access rights to a collection

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---------------|--------------------------------|---|
| POST | /rights/{collectionId}/{right} | Changes access rights: "none" (withdraws previously given rights), "read", "write", "own", of a specified user (parameter) for a specified collection (in path). Only the owner of a collection can use this call (you need to be logged in). Just one of username, email or userId needs to be provided. |

| <i>Parameters</i> | <i>Response Sample</i> |
|--|--|
| collectionID (required): string Id of the collection | { "message": "string" } |
| right (required): string Values: "none" (withdraws previously given rights), "read", "write", "own". | Response Messages |
| username : string Username of user whose rights are changed. | 400 Bad Request (no user specified) 403 Forbidden (no owner rights) 500 Internal Server Error (read/write database error) |
| membersDowngrade : boolean Downgrade access for members. Default is false. | |

Retrieve all records in a collection ordered by degree of similarity to a given record

| Method | Call | Description |
|--|--|---|
| GET | /collection/{collectionId}/similarlist | Retrieves all records from the collection specified in the path and returns an array of record objects. The records in the array are ordered by decreasing degree of similarity to the specified record. The similarity is computed using the record metadata contents. Record metadata can be available in different serializations. The format parameter defines this serialization in the records array field of the JSON response. See GET /record/{recordId}. If the format parameter is empty, only the internal record will be provided in the WITH model. |
| Parameters | | Response Sample |
| collectionId (required): string Id of the collection itemId (required): string Id of the item for which the similar list will be returned start : integer offset (default 0) count : integer | | <pre>{"entryCount": 13489,"records": [{"administrative":{"withURI":"/record/56fa7b91c743431f1cd72646","externalId":"/2048207/9999","parentResourceId":null,"access":{"isPublic":true,"acl":[]},"withCreator":null,"created":"2016/03/29","lastModified":"2016/03/29"},"collectedIn":[{"position":0,"collectionId":"56fa7b8fc743431f1cd72645"}],"usage":{"likes":0,"collected":1,"annotated":0,"viewCount":0,"tags":null},"provenance":{"provider":"Rossimoda Shoe Museum","uri":"http://www.europeana.eu/api/ANnuDzRpW/redirect?shownAt=http%3A%2F%2Fwww.europeanafashion.eu%2Frecord%2Fa%2F27f4b8d9b0b959714e2c18675ebc013a70d647485b46b878ce860fafcd490b76&provider=Europeana+Fashion&id=http%3A%2F%2Fwww.europeana.eu%2Fresolve%2Frecord%2F2048207%2F9999&profile=rich+facets"},"resourceId":null},{"provider":"Europeana</pre> |

| | |
|---|---|
| <p>count (default 10)</p> <p>format: string</p> <p>One of the following: JSON_UNKNOWN, JSONLD_UNKNOWN, XML_UNKNOWN, JSON_DNZ, XML_DNZ, JSON_YOUTUBE, "UNKNOWN", "all". If not specified, no content is returned, only basic collection fields.</p> | <pre>Fashion", "uri": null, "resourceId": null, {"provider": "Europeana", "uri": "http://www.europeana.eu/portal/record/2048207/99 99.html", "resourceId": "/2048207/9999"}, "resourceType": "C ulturalObject", "descriptiveData": {"description": {"default": ["vi tello nero, camoscio nero e accessorio dorato / black calf, black suede and golden accessory"], "it": ["vitello nero, camoscio nero e accessorio dorato / black calf, black suede and golden accessory"]}, "isShownAt": {"uri": "http://www.europeana.eu/a pi/ANnuDzRpW/redirect?shownAt=http%3A%2F%2Fwww.eur opeanafashion.eu%2Frecord%2Fa%2F27f4b8d9b0b959714e2 c18675ebc013a70d647485b46b878ce860fafcd490b76&provi der=Europeana+Fashion&id=http%3A%2F%2Fwww.european a.eu%2Fresolve%2Frecord%2F2048207%2F9999&profile=full" }, "isShownBy": {"uri": "http://repos.europeanafashion.eu/rossi moda/images/09999.JPG"}, "metadataRights": {"uri": "http://cr eativecommons.org/publicdomain/zero/1.0/"}, "rdfType": "htt p://www.europeana.eu/schemas/edm/ProvidedCHO", "count ry": {"default": ["italy"], "unknown": ["italy"]}, "dcidentifier": {"de fault": ["9999"], "unknown": ["9999"]}, "dclanguage": {"default": ["Italian"], "en": ["Italian"]}, "dctype": {"default": ["boots"], "en": ["boots"], "it": ["stivale a tubo"], "uri": ["http://thesaurus.europeanafashion.eu/thesaur us/10261"]}, "dcreator": {"default": ["not found", "non identificato (Designer)"], "en": ["not found"], "it": ["non identificato (Designer)"]}, "dccontributor": {"default": ["Rossimoda"], "unkn own": ["Rossimoda"]}, "dcformat": {"default": ["Color: black"], "en": ["Color: black"], "uri": ["http://thesaurus.europeanafashion.eu/thesaur us/10401"]}, "media": [{"Thumbnail": {"width": 0, "height": 0, "ty pe": "IMAGE", "withRights": "Creative", "url": "http://repos.euro peanafashion.eu/rossimoda/images/09999.JPG", "mediaVersi on": "Thumbnail", "originalRights": {"uri": "http://creativecomm ons.org/licenses/by-nc- nd/3.0/"}}, {"size": 0, "withUrl": "/media/byUrl?url=http://repos.e uropeanafashion.eu/rossimoda/images/09999.JPG&version= Thumbnail", "mimeType": "image/*"}, {"Original": {"width": 0, "he ight": 0, "type": "IMAGE", "withRights": "Creative", "url": "http://r epos.europeanafashion.eu/rossimoda/images/09999.JPG", "m ediaVersion": "Original", "originalRights": {"uri": "http://creative commons.org/licenses/by-nc- nd/3.0/"}}, {"size": 0, "withUrl": "/media/byUrl?url=http://repos.e uropeanafashion.eu/rossimoda/images/09999.JPG&version= Original", "mimeType": "image/*"}}, {"dbld": "56fa7b91c743431 f1cd72646"}, ...]}</pre> |
|---|---|

| | <i>Response Messages</i> |
|--|---|
| | <p>403 Forbidden (invalid collection id, no read access)</p> <p>500 Internal Server Error (cannot retrieve records from database)</p> |

4.3.4 User

Create new user

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|----------------|--|
| POST | /user/register | Creates a new user and stores at the database. |
| <i>Body Sample</i> | | <i>Response Sample</i> |
| <pre>{ "firstName": "string", "lastName": "string", "username": "string", "email": "string", "password": "string", "gender": "string", "facebookId": "string", "googleID": "string", "about": "string", "location": "string" }</pre> | | <pre>{ "email": "string", "firstName": "string", "lastName": "string", "gender": "string", "facebookId": "string", "googleID": "string", "md5Password": "string", "username": "string", "favorites": "string", "organizations": [{ "id": "string", "username": "string", "friendlyName": "string" }], "projects": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroups": [{ "id": "string", "username": "string", "friendlyName": "string" }] }</pre> |

| | |
|-----|--|
| | <pre> }, "usergroupsIds": ["string"], "adminGroups": ["string"] } </pre> |
| | Response Messages |
| 400 | Bad Request (json object describes all errors) |

User login

| Method | Call | Description |
|--|-------------|--|
| POST | /user/login | Log a user in (create a browser cookie). Some API calls do not take the user as a parameter and you need to be logged in first. You can log in with your google or Facebook id. The email parameter can be a username. |
| Body Sample | | Response Sample |
| <pre> { "email": "string", "password": "string", "googleId": "string", "facebookId": "string" } </pre> | | <pre> { "email": "string", "firstName": "string", "lastName": "string", "gender": "string", "facebookId": "string", "googleID": "string", "md5Password": "string", "username": "string", "favorites": "string", "organizations": [{ "id": "string", "username": "string", "friendlyName": "string" }], "projects": [{ "id": "string", "username": "string", "friendlyName": "string" }] } </pre> |

| | |
|-----|--|
| | <pre> }], "usergroups": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroupsIds": ["string"], "adminGroups": ["string"] } </pre> |
| | Response Messages |
| 400 | Bad Request (error status, problem description in JSON object) |

User logout

| Method | Call | Description |
|-------------------|--------------|---|
| GET | /user/logout | Browser cookie is removed, user is logged out (all session information is kept in cookie, nothing is stored on server). |
| Parameters | | Response Sample |
| | | Response Messages |
| | | 200 OK |

Check email availability

| Method | Call | Description |
|--------|----------------------|---|
| GET | /user/emailAvailable | Used when registering a new user, checks if there has been another user with the same email already stored in the database. |

| <i>Parameters</i> | <i>Response Sample</i> |
|---------------------------------|---|
| email (required): string | |
| | <i>Response Messages</i> |
| | 200 OK 400 Bad Request (not available) |

Delete a user

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|----------------|---|
| DELETE | /user/{userId} | Removes a user from the database. (This call is not allowed at the moment.) |
| <i>Parameters</i> | | <i>Response Sample</i> |
| userId (required): string Internal ID of the user. | | <pre>{ "message": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (user does not exist) |

Get user details

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|----------------|--|
| GET | /user/{userId} | Returns the complete entry of a user specified by the id provided in the path. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| userId (required): string Internal ID of the user. | | <pre>{ "email": "string", "firstName": "string", "lastName": "string", "gender": "string", "facebookId": "string", "googleId": "string", "md5Password": "string", "username": "string", "favorites": "string", "organizations": [</pre> |
| | | |

| | |
|--|---|
| | <pre>{ "id": "string", "username": "string", "friendlyName": "string" }], "projects": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroups": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroupsIds": ["string"], "adminGroups": ["string"] }</pre> |
| | Response Messages |
| | 400 Bad Request (user does not exist, exception error) |

Update a user entry

| Method | Call | Description |
|--|----------------|--|
| PUT | /user/{userId} | Updates the stored info of the user specified by the id provided in the path. |
| Parameters | | Response Sample |
| userId (required): string Internal ID of the user. Body (required): user record Content type: application/json New user entry. | | <pre>{ "email": "string", "firstName": "string", "lastName": "string", "gender": "string", "facebookId": "string", }</pre> |

| | | |
|--|--|--|
| | <pre> "googleID": "string", "md5Password": "string", "username": "string", "favorites": "string", "organizations": [{ "id": "string", "username": "string", "friendlyName": "string" }], "projects": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroups": [{ "id": "string", "username": "string", "friendlyName": "string" }], "usergroupsIds": ["string"], "adminGroups": ["string"] } </pre> | |
| | Response Messages | |
| | 400 | Bad Request (error status, problem description in JSON object) |

Send a reset password email

| Method | Call | Description |
|--------|---------------------------------------|--|
| GET | /user/resetPassword/{emailOrUserName} | Sends an email to the user provided in the path. The email contains a link to a webpage where the user can provide a new password. |

| <i>Parameters</i> | <i>Response Sample</i> |
|---|--|
| emailOrUserName (required): string | <pre>{ "message": "string" }</pre> |
| | <i>Response Messages</i> |
| | <p>400 Bad Request (invalid username or email, could not send email)</p> <p>404 Not Found (user email not found – if user had originally registered with google or Facebook account)</p> |

Get an API key

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--------------------|---------------------|---|
| POST | /user/apikey/create | Sends an API key to the stored email address of the logged in user. |
| <i>Body Sample</i> | | <i>Response Sample</i> |
| | | <pre>{ "email": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | <p>400 Bad Request (no user logged in, email already sent in past, email exception error)</p> <p>500 Internal Server Error (could not create API key)</p> |

Get the profile thumbnail of a user or group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---------------|------------------|--|
| GET | /user/{id}/photo | Returns the photoid of the thumbnail the user or group use in their profile. |

| <i>Parameters</i> | <i>Response Sample</i> |
|---|--|
| id (required): string User or group id. | <pre>{ "photoId": "string" }</pre> |
| | <i>Response Messages</i> |
| | 400 Bad Request (user or group do not exist) |

Find a user or group by name or email

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|------------------------------------|---|
| GET | /user/findByUserOrGroupNameOrEmail | Returns a JSON object with metadata for a group or a user that have the specified user/group name or email address. This call can be applied to all group types, i.e. organizations and projects. An optional functionality of this call is to check what access rights a group or user has to a collection. Just use the collectionId parameter. The access rights will be returned as an extra field named accessRights in the group or user JSON response. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| userOrGroupNameOrEmail (required): string User or group name or email. collectionId : string A collection id to check for access rights. | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (no user or group found) |

Find a user or group by name or email (autocomplete)

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|-----------------|---|
| GET | /user/listNames | This call is mainly used for autocomplete functions. It returns an array of JSON objects that contain users and groups whose names match the prefix, as well a category field with values "user" or "group". This call can be applied to all group types, i.e. organizations and projects. The onlyParents parameter filters results so that only groups or users that contain other groups or users are shown. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| prefix: string Optional prefix of user or group name. onlyParents: boolean If true filters results to groups and users that have other subgroups or sub-users. | | <pre>[{ "value": "string", "data": { "category": "string" } }]</pre> |

4.3.5 User Groups

Create a new group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|---------------|---|
| POST | /group/create | Creates a new group and stores it at the database. Every group should have an administrator. Projects and organizations are different group types, to create one of those see their respective calls. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| adminId (required): string User ID of the group administrator. adminUsername (required): string Username of the group administrator. body (required): Group Group metadata. | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"] }</pre> |

| | |
|-----|---|
| | <pre> }, "parentGroups": ["string"] } </pre> |
| | Response Messages |
| 400 | Bad Request (invalid JSON, did not specify group admin, duplicate group name or no name provided) |
| 500 | Internal Server Error (cannot save to database) |

Find a user group by name

| Method | Call | Description |
|---|-------------------|---|
| GET | /group/findByName | Returns a JSON object with metadata for the group that has the specified name. This call can be applied to all group types, i.e. organizations and projects. An optional functionality of this call is to check what access rights a group has to a collection by filling the collectionId parameter. The access rights will be returned as an extra field named accessRights in the group JSON response. |
| Parameters | | Response Sample |
| name (required): string Group name or email. collectionId : string A collection id to check for access rights. | | <pre> { "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] } </pre> |
| | | Response Messages |
| | | 400 Bad Request (no group found) |

Find subgroups of a group

| Method | Call | Description |
|---|-----------------------------------|--|
| GET | /group/descendantGroups/{groupId} | Retrieves the specified group's descendant groups in the group hierarchy. See also /group/addUserOrGroup/{groupId}. |
| Parameters | | Response Sample |
| groupId (required): string direct : boolean Only direct descendants (default is true) | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | | Response Messages |
| | | 400 Bad Request (no group found) |

Delete a group

| Method | Call | Description |
|---|------------------|---|
| DELETE | /group/{groupId} | Removes a group from the database. Only the group admin can delete the group. This call can be applied to all group types, i.e. organizations and projects. |
| Parameters | | Response Sample |
| groupId (required): string Internal ID of the user. | | <pre>{ "OK": "string" }</pre> |
| | | Response Messages |
| | | 403 Forbidden (only group admins can edit groups) |
| | | 500 Internal Server Error (cannot edit database) |

Retrieve group info

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|------------------|--|
| GET | /group/{groupId} | Retrieves attributes of a group from the database. This call can be applied to all group types, i.e. organizations and projects. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string ID of the group. | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (no group found) 500 Internal Server Error (cannot retrieve or save to database) |

Edit a group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|------------------|--|
| PUT | /group/{groupId} | Changes attributes of an existing group. You only need to provide the fields you want to change. Only the administrator of a group has the right to edit it. This call can be applied to all group types, i.e. organizations and projects. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string ID of the group. Body (required): Group | | <pre>{ "username": "string", "about": "string", "privateGroup": true,</pre> |

| | |
|---|---|
| <p>Content type: application/json</p> <p>New group metadata. You only need to provide the fields you want to change</p> | <pre>"dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | <p>Response Messages</p> |
| | <p>400 Bad Request (invalid JSON, duplicate group name or no name provided)</p> <p>403 Forbidden (only group admins can edit groups)</p> <p>500 Internal Server Error (cannot retrieve or save to database)</p> |

Add a user or another group to a group

| Method | Call | Description |
|---|---------------------------------|---|
| PUT | /group/addUserOrGroup/{groupId} | Adds a user or a group to the group with the group ID specified in the path. This call can be applied to all group types, i.e. organizations and projects. This way you can create a group hierarchy with organizations that belong to projects and different user groups that belong to organizations. |
| Parameters | | Response Sample |
| <p>Id (required): string User or group ID to add in group.</p> <p>groupId (required): string ID of the group.</p> | | <pre>{ "OK": "string" }</pre> |
| | | <p>Response Messages</p> |
| | | <p>400 Bad Request (wrong user or group id)</p> <p>403 Forbidden (only group admins can edit groups)</p> <p>500 Internal Server Error (cannot retrieve or write to database)</p> |

Remove a user from a group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|--|------------------------------------|--|
| PUT | /group/removeUserOrGroup/{groupId} | Removes a user from a group with the group ID specified in the path. This call can be applied to all group types, i.e. organizations and projects. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| Id (required): string User or group ID to add in group. groupId (required): string ID of the group. | | <pre>{ "OK": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 403 Forbidden (no rights for removal) 500 Internal Server Error (cannot retrieve or write to database) |

Request to join a group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|-----------------------|--|
| PUT | /group/join/{groupId} | Request to join the group specified by the ID in the path. The group admin will have to accept your request before you join. This call can be applied to all group types, i.e. organisations and projects. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string ID of the group. | | <pre>{ "OK": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 500 Internal Server Error (cannot retrieve or write to database) |

Leave a group

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|------------------------|--|
| PUT | /group/leave/{groupId} | Leave the group specified by the ID in the path. This call can be applied to all group types, i.e. organizations and projects. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string ID of the group. | | <pre>{ "OK": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 500 Internal Server Error (cannot retrieve or write to database) |

4.3.6 Organization and Project

Create a new organization

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|----------------------|--|
| POST | /organization/create | Creates a new organization and stores it at the database. Projects and generic groups are different group types, to create one of those see their respective calls. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| adminId (required): string User ID of the group administrator. adminUsername (required): string Username of the group administrator. body (required): Group Group metadata. | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |

| | <i>Response Messages</i> |
|-----|--|
| 400 | Bad Request(invalid JSON, did not specify group admin, duplicate group name or no name provided) |
| 500 | Internal Server Error (cannot save to database) |

Find subgroups of an organization

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|--|--|
| GET | /group/descendantOrganizations/{groupId} | Retrieves the specified group's descendant groups in the group hierarchy. See also /group/addUserOrGroup/{groupId}. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string direct : boolean Only direct descendants (default is true) | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (no group found) |

Create a new project

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---------------|-----------------|---|
| POST | /project/create | Creates a new project and stores it at the database. Organizations and generic groups are different group types, to create one of those see their respective calls. |

| <i>Parameters</i> | <i>Response Sample</i> |
|---|--|
| adminId (required): string User ID of the group administrator. adminUsername (required): string Username of the group administrator. body (required): Group Group metadata. | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], "parentGroups": ["string"] }</pre> |
| | <i>Response Messages</i> |
| | 400 Bad Request (invalid JSON, did not specify group admin, duplicate group name or no name provided) 500 Internal Server Error (cannot save to database) |

Find subgroups of a project

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|-------------------------------------|---|
| GET | /group/descendantProjects/{groupId} | Retrieves the specified project's descendant groups in the group hierarchy. See also /group/addUserOrGroup/{groupId}. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| groupId (required): string direct : boolean Only direct descendants (default is true) | | <pre>{ "username": "string", "about": "string", "privateGroup": true, "dbId": "string", "thumbnail": "string", "adminIds": ["string"], "users": ["string"], }</pre> |

| | |
|--|---|
| | "parentGroups": ["string"] } |
| | Response Messages |
| | 400 Bad Request (no group found) |

4.3.7 Media

Upload new media

| Method | Call | Description |
|---|---------------|--|
| POST | /media/create | <p>Upload a new media object (currently only supports images). This call works in two ways. You can either upload a file or provide an external url to a file. An IMPORTANT difference is that when uploading a file, the request should be a multipart/form-data. When creating from a url however, it should be a JSON body. In both cases, the fields are the same.</p> <p>You can optionally provide rights for the media, if none are provided the default value will be "UNKNOWN". Technical metadata for the new media will be automatically extracted. The returned object for this call is a list of URLs that point to the cached media object and its various representations. You can use the "original" url to create a record with the new media. The media ID is encoded in that url.</p> |
| Body Sample | | Response Sample |
| { "url": "string", "withMediaRights": "string" } | | [{ "original": "string", "tiny": "string", "square": "string", "thumbnail": "string", "medium": "string" }] |
| | | Response Messages |
| | | 400 Bad Request (invalid request) |
| | | 500 Internal Server Error (database error) |

Get metadata for a media object

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|------------------|--|
| GET | /media/{mediald} | Returns the metadata for the media object specified by the mediald. The response of the "media/create" call contains values that are url paths to this call. The response depends on the "file" parameter's value. If true, it returns the media file, if false it returns the media metadata as indicated in the response sample. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| file: boolean If true a file will be returned, if false the call returns a media metadata JSON. | | <pre>{ "width": 0, "height": 0, "type": "string", "withRights": "string", "url": "string", "size": 0, "durationSeconds": 0, "spatialResolution": 0, "bitRate": 0, "frameRate": 0, "audioChannelNumber": 0, "sampleSize": 0, "sampleRate": 0, "withUrl": "string", "mimeType": "string" }</pre> |
| | | <i>Response Messages</i> |
| | | 400 Bad Request (invalid request) 500 Internal Server Error (database error) |

Delete metadata for a media object

| <i>Method</i> | <i>Call</i> | <i>Description</i> |
|---|------------------|---|
| DELETE | /media/{mediald} | Removes the database entry for the media object specified by the url parameter. |
| <i>Parameters</i> | | <i>Response Sample</i> |
| mediald: string The id of the media | | <pre>{ "message": "string" }</pre> |

| | <i>Response Messages</i> | |
|--|--------------------------|--|
| | 400 | Bad Request (invalid request) |
| | 500 | Internal Server Error (database error) |

4.4 INTERACTIVE DOCUMENTATION USING SWAGGER

Swagger is a framework for representing a RESTful API through a formal specification and then using this with a set of tools it offers. It can automatically extract the API schema from many programming languages, create interactive documentations and offers many other functionalities. In the WITH case, the API schema is specified top down using the provided editor which uses the YAML language to script the definitions. This definition is then translated to a JSON structure automatically from the editor and used in two different user interfaces in the developers page of the application.

4.4.1 API Keys

API keys are special parameters in the requests that authenticate the origin of the call, therefore each call to the WITH API needs an API key. Access to the API is open, so any registered user can request a key from the developers page (see next section). Clicking on the “Request an api key” button will result to an email being sent with the API key. If a user already has an API key but forgot it, they will need to send an email in order to get a new one.

API keys can be provided to the API in three ways. First, they can be provided with a simple “apikey” parameter in each request. Second, they can be stored in a session cookie which the back-end sees and extracts from. These methods can be used by developers or for back-end service communications. Neither of these methods is very secure however, so the WITH front-end uses a different way to communicate with the back-end.

The WITH application (its UI and subsequently the Technical Space front-end), connects using an API key to the back-end. This key should not really be made available publicly, so a custom transmission scheme is employed that encodes this key and obfuscates its extraction. The back-end decodes the key, authenticates it, and then re-encodes it in a different obfuscation scheme. Developers who wish to create third party web applications to be distributed to users, will have access to certain JavaScript functions that perform this encoding, upon request and verification.

4.4.2 Swagger UI

The main interface is using the Swagger UI responsive theme (Figure 4). This is chosen as the main interface as the design is considered to be more intuitive to an unfamiliar user, and also because of the good overview the left hand menu gives for the whole API options. The secondary API-LITE (Figure 5) interface is the standard Swagger UI that is provided in the swagger website. Both of those have some custom modifications, such as buttons to request an API key. The biggest functionality these interfaces offer is the option to test the calls with sample requests directly from the documentation page. However, depending on the parameters a call may require, this may need additional configuration.

We present an example simple search call and its representation in the Swagger user interfaces. The following image shows the main documentation page, and the one after shows the lite version. Note that these pages offer the same information and functionalities, however the first one has a better layout for exploring the API documentation, whereas the second is faster for testing out calls.

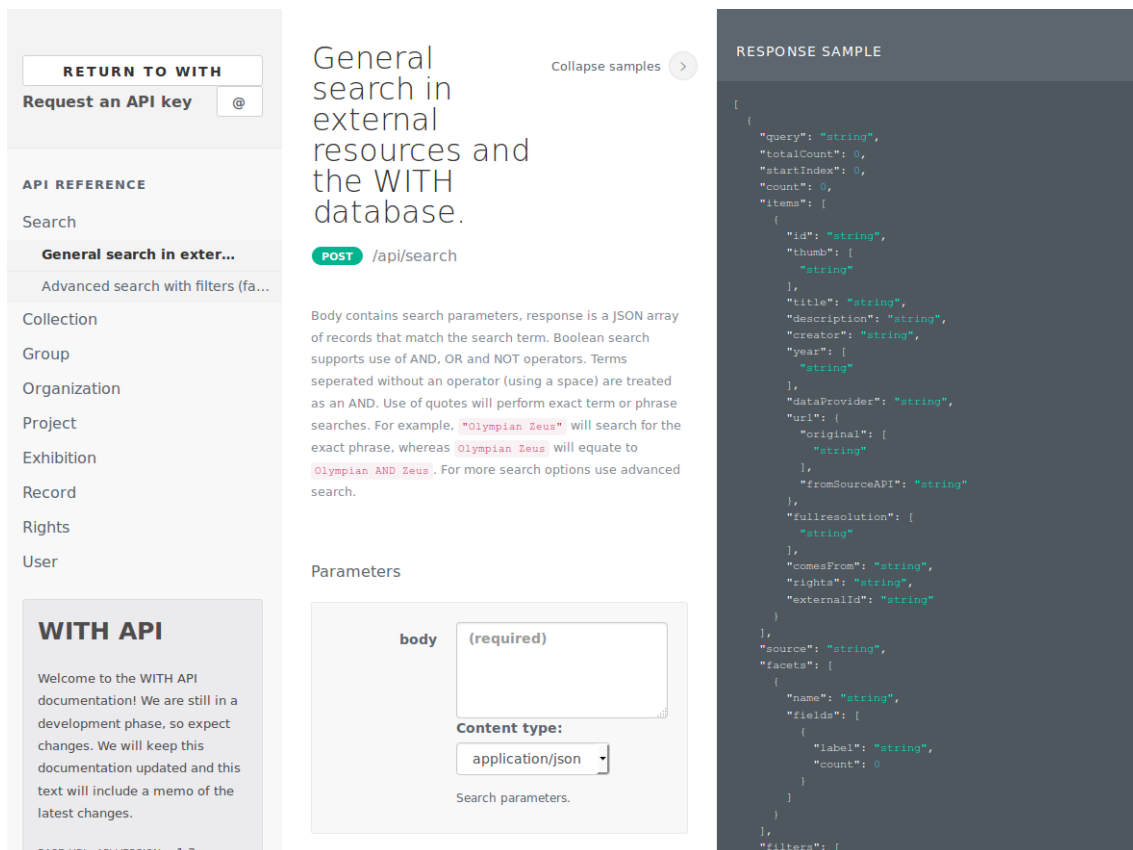



Figure 4. API swagger documentation

 **swagger**

Return to WITH

WITH API

Welcome to the WITH API documentation! We are still in a development phase, so expect changes. We went through some major updates but many changes will follow.

Search

Show/Hide | List Operations | Expand Operations

POST

/api/search

General search in external resources and the WITH database.

POST

/api/advancedsearch

Advanced search with filters (faceted sarch).

| | | | |
|--------------|-----------|-----------------|-------------------|
| Group | Show/Hide | List Operations | Expand Operations |
| Organization | Show/Hide | List Operations | Expand Operations |
| Project | Show/Hide | List Operations | Expand Operations |
| Collection | Show/Hide | List Operations | Expand Operations |
| Exhibition | Show/Hide | List Operations | Expand Operations |
| Record | Show/Hide | List Operations | Expand Operations |
| Rights | Show/Hide | List Operations | Expand Operations |
| User | Show/Hide | List Operations | Expand Operations |

[BASE URL: , API VERSION: v1.3]

Figure 5. API Swagger documentation – Lite view

In the main documentation page, on the left hand side there is a menu where all API calls are sorted by category. Calls that cover two categories, for example the one that adds records to a collection, can be duplicated. By clicking on a category and selecting a call, the middle of the page displays a description of the call and all the path or header parameters it can take. By clicking on the “Show samples” link in the top right, the right hand menu (Figure 6) displays four options to show the following:

- Response sample: The JSON object of the response to this call.
- Response schema: Details about the response object and its fields.
- Body sample: The JSON body of the call, if it has one.
- Body schema: Details about the body object and its fields.

The image shows a sidebar from a Swagger API documentation tool. It is divided into several sections: 'RESPONSE SAMPLE', 'RESPONSE SCHEMA', 'BODY SAMPLE', 'BODY SCHEMA', and a list of parameters. The 'BODY SAMPLE' section contains a JSON object with fields 'searchTerm', 'page', 'pageSize', and 'source'. The 'BODY SCHEMA' section lists these fields with their respective data types and optional status. The parameters section lists 'searchTerm', 'page', 'pageSize', and 'source' with their data types and optional status.

RESPONSE SAMPLE

RESPONSE SCHEMA

BODY SAMPLE

```
{
  "searchTerm": "string",
  "page": 0,
  "pageSize": 0,
  "source": [
    "string"
  ]
}
```

BODY SCHEMA

searchTerm
string (optional)

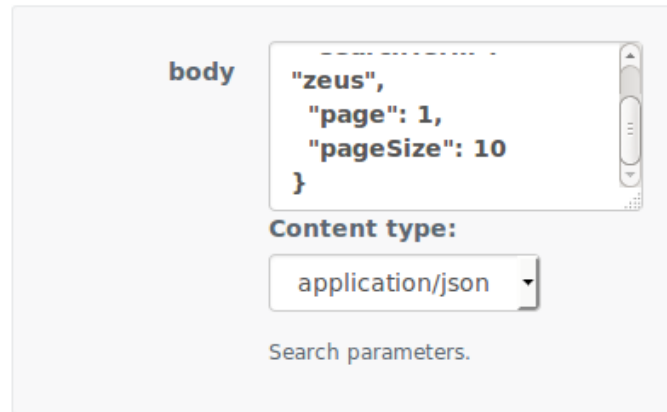
page
integer (optional)

pageSize
integer (optional)

source
array[string] (optional)

Figure 6. Swagger "Show Samples" sidebar

Parameters



The image shows a Swagger UI 'Parameters' section. On the left, the word 'body' is displayed. To its right is a text area containing a JSON object: `"zeus",`
`"page": 1,`
`"pageSize": 10`
`}`. Below the text area, the label 'Content type:' is followed by a dropdown menu currently showing 'application/json'. At the bottom of the section is a text input field with the placeholder text 'Search parameters.'

Figure 7. Swagger parameters box for a call

By clicking on the “Body sample” container, the parameters box (Figure 7) in the bottom middle of the page fills with this sample, as seen in the following images. After setting the values, the call can be tested and the results will show in a pop up container in the page (Figure 8). In the Lite API page this will also work, however due to the different layout all this information is shown right under the call in the same menu, the description, the models, the schema, even the test call response (Figure 9).

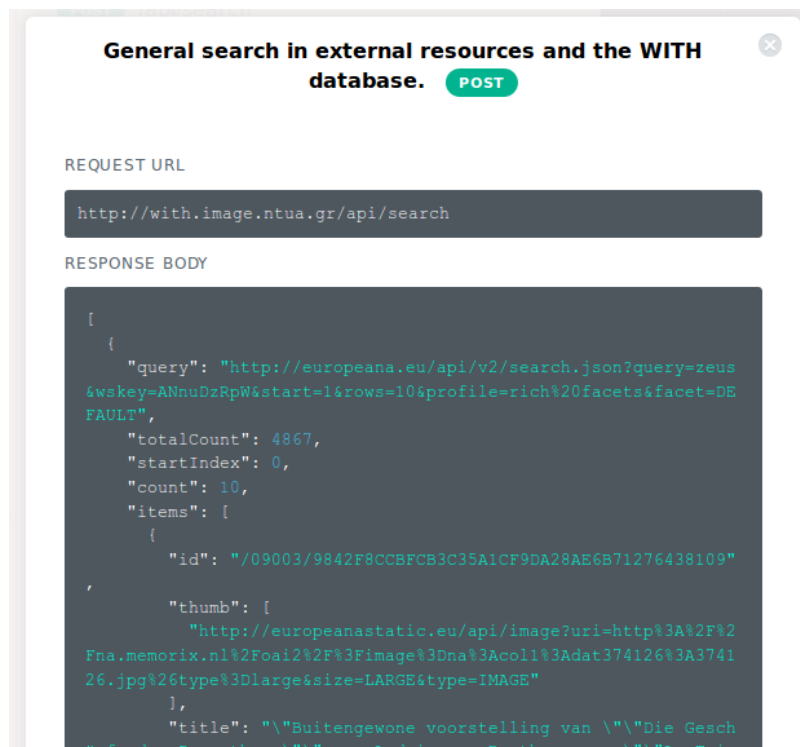


Figure 8. Swagger API call results from test

Search

Show/Hide | List Operations | Expand Operations

POST /api/search

General search in external resources and the WITH database.

Implementation Notes

Body contains search parameters, response is a JSON array of records that match the search term. Boolean search supports use of AND, OR and NOT operators. Terms separated without an operator (using a space) are treated as an AND. Use of quotes will perform exact term or phrase searches. For example, "Olympian Zeus" will search for the exact phrase, whereas Olympian Zeus will equate to Olympian AND Zeus. For more search options use advanced search (coming soon).

Response Class (Status 200)

Model | Model Schema

```

{
  "sourceId": "string",
  "sourceUrl": "string",
  "exhibition": {
    "annotation": "string",
    "audioUrl": "string",
    "videoUrl": "string"
  },
  "position": 0
}

```

Response Content Type

application/json

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|-----------|--|--------------------|----------------|--|
| body | <pre> { "searchTerm": "zeus", "page": 1, "pageSize": 10 } </pre> | Search parameters. | body | <div>Model Model Schema</div> <pre> { "searchTerm": "string", "page": 0, "pageSize": 0, "source": { "string" } } </pre> <div>Click to set as parameter value</div> |

Response Messages

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|-------------|----------------|---------|
| 403 | Bad request | | |

Try it out!

Hide Response

Curl

```

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d '{
  "searchTerm": "zeus",
  "page": 1,
  "pageSize": 10
}' http://with.image.ntua.gr/api/search

```

Request URL

```

http://with.image.ntua.gr/api/search

```

Response Body

```

{
  "query": "http://europeana.eu/api/v2/search.json?query=zeus&wskey=ANnuDzRpW&start=1&rows=10&profile=rich%20fa",
  "totalCount": 4867,
  "startIndex": 0,
  "count": 10,
  "items": [
    {
      "id": "/09003/9842F8CCBFCB3C35A1CF9DA28AE6B71276438109",
      "thumb": [
        "http://europeanastatic.eu/api/image?uri=http%3A%2F%2Fna.memorix.nl%2Ffoai2%2F%3Fimage%3Dna%3Acol1%3Adat",
        "http://europeanastatic.eu/api/image?uri=http%3A%2F%2Fna.memorix.nl%2Ffoai2%2F%3Fimage%3Dna%3Acol1%3Adat"
      ],
      "title": "\"Buitengewone voorstelling van \"\"Die Geschöpfe des Prometheus\"\" van Ludwig van Beethoven en",
      "description": "",
      "creator": "",
      "year": [
        "1939"
      ],
      "dataProvider": "Nationaal Archief",
      "url": {
        "original": [
          "http://europeanastatic.eu/api/image?uri=http%3A%2F%2Fna.memorix.nl%2Ffoai2%2F%3Fimage%3Dna%3Acol1%3Adat"
        ]
      }
    }
  ]
}

```

Response Code

200

Response Headers

```

{
  "date": "Thu, 15 Oct 2015 12:50:23 GMT",
  "content-type": "application/json; charset=utf-8",
  "content-length": "101157",
  "keep-alive": "timeout=5, max=99",
  "connection": "Keep-Alive"
}

```

Figure 9. Testing API calls in Swagger Lite UI

5 INTEROPERABILITY WITH THE JPSEARCH FRAMEWORK

5.1 BACKGROUND

Managing image data poses multiple challenges. Several copies of the same images may be spread over several systems. When images are moved from one platform to another, metadata – such as annotations or copyright information – is not always preserved. Often, this is due to the usage of different metadata formats or usage of non-compliant import and export schemes.

Searching digital images is not a trivial task either. One cause is inherent to the lack of consistency in usage of metadata schemas. This makes addressing specific metadata elements problematic. A second reason is the so-called semantic gap. Since not all images are textually annotated, methods have been explored for querying images by content. However, most common query languages do not support these novel techniques. Finally, accessing content of online repositories is not organized in a consistent way. Many of these repositories provide proprietary APIs, which differ from repository to repository and are often too restrictive to handle more advanced queries.

JPSearch is a set of standards that aim to address interoperability in image search and retrieval systems. JPSearch defines an abstract image search and retrieval framework. Interfaces and protocols for data exchange between the components of this architecture are standardized, with minimal restrictions on how these components perform their respective tasks. The use and reuse of metadata and associated metadata schemas is thus facilitated. A common query language is also defined to enable search over distributed repositories. Finally, an interchange format is specified to allow users to easily import and export their data and metadata among different applications and devices¹⁶.

The JPSearch standards are defined by the Joint Photographic Experts Group (JPEG). JPEG is a joint working group of the International Standardization Organization (ISO) and the International Electrotechnical Commission (IEC). It resides under JTC1, which is the ISO/IEC Joint Technical Committee for Information Technology. More specifically, the JPEG committee is Working Group 1 (WG1), Coding of Still Pictures, of JTC 1's subcommittee 29 (SC29), Coding of Audio, Picture, Multimedia and Hypermedia Information. The word “Joint” in JPEG refers to an additional collaboration with the International Telecommunication Union (ITU).

Although JPEG is mainly known for its image coding standards, the JPEG committee has increased its efforts in providing more system level support for its suite of standards. One of these efforts resulted in the JPSearch specification. The specification is composed of different components addressing the system framework (Part 1), schema and ontology building blocks (Part 2), the query format (Part 3), the file format for metadata embedded in image data (Part 4), the data interchange format for image repositories (Part 5) and finally the reference software (Part 6). In-depth information on JPSearch can be found on www.jpeg.org.

16 Frederik Temmermans, Frederic Dufaux, Peter Schelkens, JPSearch: Metadata Interoperability During Image Exchange, IEEE Signal Processing Magazine, Issue 5, Volume 29, pp 134 – 139, IEEE, 2012

On one hand, Europeana Space aims to increase the awareness of standards in the cultural heritage community. On the other hand, it is important to bring feedback from the community back to the JPEG committee in order to help them to shape future standards that fit the needs of the cultural heritage sector when dealing with still images. This is done by participating in workshops and representation and involvement in the JPEG activities.

This chapter focuses on the guidelines and actions taken to achieve interoperability of cultural heritage image repositories with the JPSearch framework. In addition, it describes an open source implementation of the JPSearch API that is provided as a reference. This implementation will also be provided to participants of the Photography pilot hackathon.

5.2 ACHIEVING INTEROPERABILITY WITH THE JPSEARCH FRAMEWORK

5.2.1 Embedding metadata in JPEG images

A main cause of metadata loss is that often metadata is decoupled from the image file itself, i.e. the metadata is stored and managed independently from the image. When the image is downloaded and moved to another repository, the metadata is not retained. JPSearch Part 4 – File Format – specifies a file format for embedding any kind of metadata in a JPEG or JPEG 2000 image. By embedding the metadata into the image itself, the metadata stays attached with the image, even when it is moved between applications or platforms. The file format is designed to be backward compatible with previously launched JPEG image coding standards. As a consequence, the images will be readable by any decoder. The file format allows multiple metadata instances to be embedded into a single image, even if these have different schemas.

Currently, the file format requires at least one embedded instance of the JPSearch Core metadata. During the Europeana Space IPR Workshop in Coventry this was considered as a downside by the community, since it requires supporting yet another additional schema. However, not including a JPSearch Core schema should not necessarily restrict compatibility, since translation rules can be registered using the registration authority, as discussed further on in this document. Therefore, a proposal was made during the 69th JPEG meeting in June 2015 in Warsaw to drop this requirement in addition to other improvements to the file format. The proposal was well received and will be followed up during upcoming JPEG meetings.

At the time of writing, JPSearch does not provide means to protect metadata through encryption. During the Europeana Space IPR Workshop in Coventry, it was discussed that this an important feature, especially when metadata stays attached to the image. This feature is however a requirement of the new Privacy and Security work item of the JPEG committee that was officially launched during a workshop on 13 October 2015 in Brussels. Peter Schelkens and Frederik Temmermans from iMinds represented Europeana Space in the organizing committee of the workshop and Charlotte Waelde from UNEXE and Fred Truyen from KU Leuven attended to give pertinent presentations during the workshop.

5.2.2 Metadata mapping and registration

In practice several metadata schemas are used alongside each other, often with overlapping terms. To achieve interoperability, systems require mechanisms for translating metadata from one schema into another. To that end, JPSearch defines the Translation Rules Declaration Language (TRDL). This is an XML-based language that provides means for translating metadata to and from the JPSearch Core Schema into equivalent information of an external (XML serializable) schema. The TRDL focuses on mappings at the structural and syntactic level. A transformation model specifies multiple rules with source and target format.

One field can be split or multiple fields can be combined during the translation. For more complex conversions, such as splits or merges of certain fields, the TRDL provides the ability to declare regular expressions over the content of elements or attributes.

Metadata schemas and translation rules can be registered via the JPsearch Registration Authority (RA). The JPsearch RA is an official body designated by ISO hosted by the Distributed Multimedia Applications Group (DMAG) of the Universitat Politècnica de Catalunya - BarcelonaTech (UPC) (<http://dmag.ac.upc.edu/jpsearch-ra>).

In context of this project the Europeana Semantic Elements (ESE) schema was registered with the JPsearch RA. ESE adopts Dublin Core (DC) element set as a basis. The Dublin Core element set maps to the JPsearch Core metadata schema in order to provide interoperability with JPsearch queries. Recently, Europeana has moved from ESE to the RDF (Resource Description Framework) based Europeana Data Model (EDM). More information on dealing with RDF or linked data is provided in the next section.

5.2.3 Linked data

Using predefined metadata schema, tags or key-value pairs entails a specific drawback, these annotations are not necessarily universally interpreted in the same way. The annotations may be language dependent or sensitive to differences in interpretation. This is one of the reasons why Europeana has moved from ESE to EDM. JPEG also acknowledged this and therefore recently introduced the JPEG Ontology for Still Image Descriptions (JPonto). The main goal is to provide a simple and uniform way of annotating JPEG images with metadata compliant to the Linked Data principles. JPonto specifies how RDF metadata annotations should be embedded in JPEG or JPEG 2000 files and provides a core ontology for describing images. Since ontologies are often very specialized and domain specific, as is the case for EDM, mechanisms are foreseen to extend JPonto core via the registration authority.

5.3 JPSEARCH API

This section describes the JPsearch API. The JPsearch API intends to give client applications access to image repositories in an interoperable way. The API provides methods for accessing variations of an individual image, image collections and metadata. A lightweight open source reference implementation is provided that can easily complement existing repository interfaces.

5.3.1 Basic concepts

The JPsearch API defines a set of queries expressed in the Uniform Resource Identifier (URI) syntax, following the RFC 3986 syntax, as shown below:

scheme '://' authority '/' path ['?' query] ['#' fragment]

The API focuses solely on the “query” part of the former definition. The query part is a sequence of key value pairs separated with an “&” character. The keys are referred to as arguments or query options. Query options reserved by the JPsearch API are referred to as “system query options”. These query options can have several types including numbers, intervals, strings, timestamps among others.

The JPSearch API does not impose any restrictions on the remaining part of the URI, i.e. the scheme, authority, path and fragments. Furthermore, additional query options can be specified, as long as they do not infer with the system query options. In case there is an overlap with system query options, a prefix can be specified in the capability descriptions.

The capability description is a required resource that should be served by any JPSearch-server. It specifies the available functionalities, restrictions and additional information such as authentication. System query option can be enabled or disabled and default values can be overwritten. The capability description is requested in an initial interaction between a client and server. It informs the client about what queries can be sent and how they should be formatted for the respective repository.

A JPSearch-client requests resources from a JPSearch-server. In addition to the capability description, the main types of resources are images, metadata and collections. The following sections provide more information on requesting these resources. However, the provided information introduces basic concepts and principles but is not all-embracing. More detailed information can be found in the specification of the standard. In addition, the information included with the software specifies available functionality and usage of the options.

5.3.2 Image resources

The JPSearch API defines a set of requests at the level of an individual image. In practice, this means that the same image can be provided at different sizes, different qualities, including or stripped metadata and so on. The return type of these requests is a JPEG or JPEG 2000 image file. The original image can be requested by its resource identifier, without specifying any system query options. Various versions of the original image can be requested by specifying system query options. An implementation of the API should not necessarily support all system query options. Enabled and disabled options are determined by the capability description. The requestable versions of an image can either be static, i.e. pre-generated, or they can be generated dynamically on request. This implementation generates the versions dynamically. The versions are generated using ImageMagick (<http://www.imagemagick.org>). ImageMagick is an open source (Apache 2.0 license) software package to create, edit, compose or convert images. It can read and write images in a variety of formats including JPEG and JPEG 2000. In addition to ImageMagick, jhead is used for metadata manipulation (<http://www.sentex.net/~mwandel/jhead/>). The server side code of is written in PHP.

Here is a list of system query options at the image level:

- **crop** Specifies whether the returned image should be cropped and at what aspect ratio.
- **includemd** Specifies whether the embedded metadata should be included or discarded.
- **maxw** Specifies the maximum width of the returned image.
- **maxh** Specifies the maximum height of the returned image.
- **quality** Specifies the quality of the returned image as a value between 0 and 100 where 0 is the lowest quality and 100 is the highest quality.
- **roff** Region offset. Used in combination with rsize to request a rectangle region of interest.

- **rsiz** Region size. Used in combination with roff to request a rectangle region of interest.
- **scale** Scale of the returned image.
- **thumb** Specifies whether the image should be returned as a thumbnail. When a thumbnail is requested, maxw and maxh are set to 256, metadata to discard, crop to square (and quality to 50). These values are overwritten when one of these arguments is specified.

5.3.3 Image metadata

Some applications need to present metadata of an image without presenting the image itself. Therefore, metadata can be requested separately from the image by specifying an image URI in combination with the metadata system query option. Metadata fields can be selected of the JPSearch Core Metadata schema, as defined in ISO/IEC 24800-2:2011. The value of the metadata argument is a comma-separated list of the requested metadata fields represented by their name or XPath expression. For example, the following request:

```
http://www.repository.org/image.jpg?metadata=
Title,Creators,GPSPositioning/@latitude,GPSPositioning, RightsDescription/Description
```

will return the following fields:

- Title: content of the title field
- Creators: the GivenName and FamilyName of the creators, space separated
- GPSPositioning/@latitude: latitude attribute value of the GPS localization
- GPSPositioning: complete GPS positioning
- RightsDescription/Description: content of the Description field of the RightsDescription element

Alternatively, when the value is set to all, the complete JPCore metadata is returned. In this case, the return format is XML, i.e. JPCore metadata is returned according to the ISO/IEC 24800-2:2011 specification. Finally, JPOnto compliant RDF descriptions can be requested with the parameter set to description.

5.3.4 Collections

A collection is a set of images identified with a URI. In general, a collection is a subset of the images served by a repository. The return type of a collection is a JSON (application/json) file listing the resources of the images in the collection.

Queries at the level of a collection can be distinguished between metadata agnostic system query options and JPSearch metadata based conditional queries. Metadata agnostic query options provide arguments for common queries such as location, size, colour, time, type and free text. Conditional queries support common operators such as equals, greater than, and, or, not, etc. on JPCore metadata fields. For both types of queries, the implementation makes use of a SQL database that contains a relational representation of the metadata of the images in a collection. A JPSearch API query is translated to a SQL query on this database. The results are then in their turn translated in the JPSearch API output format for representing a collection.

6 CONCLUSIONS

This document has described the Technical Space application data model and its implementation and has fully documented the platform's API. Additional details on the operation of the system can be found in deliverable D2.3 that documents the Europeana Space infrastructure. The external APIs combination feature is outlined, as the major facilitator towards efficient discovery and re-use of available CH content. The development of the API is presented through a revision of the requirements phase and its first two major releases. An exhaustive list of API calls covering version 2 is documented. Finally, guidelines and actions taken to achieve interoperability with the JPSearch framework for image search and retrieval are discussed.

It is important to note that the development of the platform is ongoing, following the activities of the project and taking advantage of the feedback gathered from the pilot teams and the usage of the API by third-party developers. In the same time, other initiatives and projects take advantage of the platform's functionality to enable their strategies towards collecting, visualizing and making available their CH content, thus providing new channels of feedback and guiding the development towards intuitive and innovative functionalities. The online documentation page offers the latest release information, instructions for using the platform, and the latest news about the Technical Space APIs.

To give an interesting example of the platform's evolution, NTUA is currently working on providing the infrastructure to create and operate crowdsourcing workflows. These aim to present and validate CH resources documentation and collect annotations from users in a formal, semantic web based approach that allows for the formal recording of all information and its provenance. The Europeana Sounds project aims to implement such a workflow, using the Pundit tool that is integrated in the platform, to crowdsource annotations for the project's CH resources in their efforts to enrich and validate the metadata with the public's participation.