

**EGI-Engage**

Data repository for DARIAH

D6.2

|  |  |
| --- | --- |
| **Date** | 18 December 2015 |
| **Activity** | SA2 Knowledge Commons |
| **Lead Partner** | INFN |
| **Document Status** | DRAFT |
| **Document Link** | <https://documents.egi.eu/public/ShowDocument?docid=2654>  |

Abstract

This deliverable describes the outcome of the “Storing and Accessing DARIAH contents on EGI (SADE)” pilot application of the DARIAH Competence Centre of the EGI-Engage project. The activity developed and deployed a digital repository based on gLibrary framework and the EGI Federated cloud and grid infrastructure, in order to address data cataloguing requirements of the DARIAH community. The presented work was developed in EGI by the EGI DARIAH Competence Centre of the EGI‑Engage project.

**COPYRIGHT NOTICE**



This work by Parties of the EGI-Engage Consortium is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/). The EGI-Engage project is co-funded by the European Union Horizon 2020 programme under grant number 654142.

**DELIVERY SLIP**

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Name*** | ***Partner/Activity*** | ***Date*** |
| **From:** | Giuseppe La Rocca, Roberto Barbera | EGI.eu / INFN | 17/12/2015 |
| **Moderated by:** |  |  |  |
| **Reviewed by** |  |  |  |
| **Approved by:** |  |  |  |

**DOCUMENT LOG**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Issue*** | ***Date*** | ***Comment*** | ***Author/Partner*** |
| **V.0.1** | 07/12/2015 | Document creation | G. La Rocca / EGI.eu, INFN |
| **V0.2** | 15/12/2015 | Updated version | G. La Rocca, Antonio Calanducci /EGI.eu, INFN |
| **V0.3** | 17/12/2015 | Minor update | R. Barbera / INFN |
| **V1** | 17/12/2015 | Draft for external review | G. Sipos / EGI.eu-SZTAKI |
| **V1.1** | 18/12/2015 | Final version | G. La Rocca /EGI.eu, INFN |

**TERMINOLOGY**

A complete project glossary is provided at the following page: <http://www.egi.eu/about/glossary/>

**Contents**

[1 Introduction 6](#_Toc438204924)

[2 Service architecture 8](#_Toc438204925)

[2.1 The gLibrary Digital Repository Service 8](#_Toc438204926)

[2.2 gLibrary architecture with Science Gateways 10](#_Toc438204927)

[2.3 The Semantic Search Engine High-Level Service architecture 11](#_Toc438204928)

[2.3.1 Metadata Harvester 12](#_Toc438204929)

[2.3.2 Semantic-Web Enricher 12](#_Toc438204930)

[2.3.3 Search Engine 13](#_Toc438204931)

[2.4 Integration and dependencies 16](#_Toc438204932)

[3 Requirements 17](#_Toc438204933)

[3.1 Functional Requirements 17](#_Toc438204934)

[3.2 Non-functional Requirements 17](#_Toc438204935)

[4 Feedback on satisfaction 17](#_Toc438204936)

[5 Future plans 17](#_Toc438204937)

[Appendix I. gLibrary 2.0 API Docs 19](#_Toc438204938)

[Server endpoint 19](#_Toc438204939)

[Authentication 19](#_Toc438204940)

[Login 19](#_Toc438204941)

[User creation 19](#_Toc438204942)

[Authorization 20](#_Toc438204943)

[ACLs 20](#_Toc438204944)

[Repositories 20](#_Toc438204945)

[List of all the repositories hosted on the server 21](#_Toc438204946)

[Create a new repository 21](#_Toc438204947)

[Collections 23](#_Toc438204948)

[Create a new collection 23](#_Toc438204949)

[Import data from an existing relational database 24](#_Toc438204950)

[List all the collections of a repository 24](#_Toc438204951)

[Retrieve the schema of a collection 25](#_Toc438204952)

[Delete a collection 27](#_Toc438204953)

[Items (previously entries) 27](#_Toc438204954)

[Item creation 27](#_Toc438204955)

[Item listing 27](#_Toc438204956)

[Item detail 28](#_Toc438204957)

[Item deletion 28](#_Toc438204958)

[Item update 28](#_Toc438204959)

[Queries with filters 28](#_Toc438204960)

[Replicas 28](#_Toc438204961)

[Replica creation 29](#_Toc438204962)

[Retrieve all the replicas of the given *item\_id* 29](#_Toc438204963)

[Download a given replica 29](#_Toc438204964)

[Upload a replica 29](#_Toc438204965)

[Delete a replica 30](#_Toc438204966)

[Relations 30](#_Toc438204967)

[Retrieve related items 30](#_Toc438204968)

[Contributors 30](#_Toc438204969)

**Executive summary**

This deliverable reports the status of the “Storing and Accessing DARIAH contents on EGI (SADE)” pilot application of the DARIAH Competence Centre (CC) of the EGI-Engage project. This pilot application relies on several services from the National Grid Initiatives (NGIs). These services have been further extended and improved by the Competence Centre to better address the requirements coming from the DARIAH CC and to simplify and customise access to the EGI Federated Cloud Infrastructure for the digital humanities (DH) community.

The outline of this deliverable is as follows: first we provide a short introduction of the goal of the pilot application. Then the high-level architectures of the implemented frameworks are described along with the technological building blocks and their inter-dependencies. Finally, future development activities are described in the final section. The document closes with an appendix that includes the documentation of the online system.

# Introduction

In the last decade excellent Science has become more and more possible with e-Infrastructures. However, although various research domains, such as Medicine, Chemistry, and Physics, have already made extensive use of e-Infrastructure services, others, such as Arts and Humanities are not yet exploiting these facilities at the same pace. To bridge this gap, the EGI DARIAH Competence Centre (CC) project[[1]](#footnote-1) aims to provide a wider and more efficient access to, and use of e-Infrastructures, particularly the Federated Cloud facility offered by EGI. The CC aims to make the EGI Federated Cloud more convenient to use for members of the European digital research infrastructure for the Arts and Humanities[[2]](#footnote-2).

In this document we report the status of the “Storing and Accessing DARIAH contents on EGI (SADE)” pilot application that is part of the CC activities. SADE is one of the two mini-projects within the CC and aims to raise awareness of the possible benefits in using e-Infrastructure and e-Science technology in the A&H scientific domain. The overall goal of the SADE mini-project is to develop a multi-channel digital repository of DARIAH contents on top of the EGI Cloud Storages and, on the example of the use case of electronic lexicography and lexical resources, and demonstrate the liaison with semantic web technologies projects and research infrastructures.

With regards to the storage back-end, cloud technologies are mature enough to improve the reliability and the availability of data. Clouds can provide the possibility to dynamic data replication among computing centres in a user-transparent way, preventing data loss due to localized disasters.

For this pilot application the DARIAH contents have been provided by the Austrian Academy of Sciences[[3]](#footnote-3) (AAS), one of the leading Austrian research institutions with a very long-running experience and interest in the Arts and Humanities domain. The AAS datasets represent the work on a 100+ years old collection on Bavarian dialects within the Austrian-Hungarian monarchy from the beginnings of German language to nowadays. Several data types are taken into account: text, multimedia (images, audio files etc.), URIs; as well as primary collection data, interpreted data, secondary background data and geo-data with different license opportunities.

The Italian National Institute for Nuclear Physics (INFN) setup two different frameworks in the pilot to demonstrate capabilities relating to the two key aspects of digital repositories:

1. Building customised and highly-available digital repositories.
2. Discover and correlate content in geographically distributed digital repositories.

The first setup of the pilot provides to a lightweight and sophisticate tool for members of the DH community to organize and manage digital repositories on top of the EGI Federated Cloud. This system is based on the gLibrary system. The second setup offers a semantic search engine (SSE) for the DH community that can be used for keyword-based content discovery.

The remaining of this document presents the architecture of the two frameworks that have been setup and tailored for the DH community.

The following tables provide a summary of the key aspects of the two systems:

|  |  |
| --- | --- |
| **Tool name** | gLibrary, the INFN Digital Repository System |
| **Tool url** | Current link: <https://glibrary.ct.infn.it>Testing endpoint: <https://glibrary.ct.infn.it:3500>  |
| **Tool wiki page** | <https://csgf.readthedocs.org/en/latest/glibrary/docs/index.html> |
| **Description** | A framework to create and manage repositories of digital assets on grid, cloud and local storage |
| **Customer of the tool** | DARIAH CC |
| **User of the service** | Research groups; individual A&H researchers |
| **User Documentation**  | <https://csgf.readthedocs.org/en/latest/glibrary/docs/index.html> |
| **Technical Documentation**  | <https://csgf.readthedocs.org/en/latest/glibrary/docs/index.html><http://csgf.readthedocs.org/en/latest/glibrary/docs/glibrary2.html>(APIs documentation) |
| **Product team** | INFN |
| **License** | Released under the Apache License 2.0 |
| **Source code** | <https://github.com/csgf/glibrary> (source code) |

|  |  |
| --- | --- |
| **Tool name** | The Semantic Search Engine |
| **Tool url** | Current link: <http://csgf.egi.eu/dariah-sse-parallel>  |
| **Tool wiki page** | <https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html> |
| **Description** | A framework conceived to demonstrate the potential of Open Access Data infrastructures coupled with semantic web technologies to address the issues of data discovery and correlation |
| **Customer of the tool** | DARIAH CC |
| **User of the service** | Research groups; individual A&H researchers |
| **User Documentation**  | <https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html> (PSSE)<http://csgf.readthedocs.org/en/latest/semantic-search-portlet/docs/index.html> (SSE)<http://csgf.readthedocs.org/en/latest/semantic-search-api/docs/index.html> (API SSE) |
| **Technical Documentation**  | <https://csgf.readthedocs.org/en/latest/parallel-semantic-search-portlet/docs/index.html>  |
| **Product team** | INFN |
| **License** | Released under the Apache License 2.0 |
| **Source code** | <https://github.com/csgf/parallel-semantic-search-portlet> (PSSE)<https://github.com/csgf/semantic-search-portlet> (SSE)<https://github.com/csgf/semantic-search-api> (API) |

# Service architecture

In this section we describe the high-level service architecture of the two frameworks, along with their dependencies, API and software requirements, that have been customized for the Arts and Humanities scientific community. The first one is gLibrary, the digital repository system developed by INFN to offer an easy-to-use service and a powerful system to handle digital assets. The second one is the Semantic Search Engine (SSE), the framework conceived to demonstrate the potential of information coupled with semantic web technologies to address the issues of data discovery and correlation. With the SSE user can search keywords in the e-Infrastructure Knowledge Base, in more than 100 languages across more than 30 million resources contained in the thousands of semantically enriched Open Access Document Repositories (OADRs) and Data Repositories (DRs). Search results are ranked according to the Ranking Web of Repositories[[4]](#footnote-4). For supporting the requirements of the Arts and Humanities scientific community a parallelised version of the SSE has been configured to simultaneously search, using different Java threads, across the above e-Infrastructure Knowledge Base and other platforms.

## The gLibrary Digital Repository Service

gLibrary is the digital repository system developed by INFN. It has been used in different national and international EU-funded projects in fields such as Humanities, Earth Science and High Energy Physics. In the past months, gLibrary has evolved from a system based on Grid Storage and metadata only to a more general service, capable of handling repositories on both Grid and Cloud Storages with metadata and data records stored in both relational and/or non-relational databases. The new version of the system, called gLibrary 2.0, offers access both to existing data repositories and the creation of new ones via a simple REST API.

For the sake of completeness and to help reader’s comprehension below we report some terminologies used in gLibrary lingo.

A **repository** is a virtual container of one or more data **collections**.

A **collection** provides access to data records stored on a relational DB table or to a non-relational (NoSQL) DB collection. Currently gLibrary supports MySQL, PostgreSQL, Oracle and MongoDB.

Each repository can group together one of more collections, providing a virtual and uniform interface to data tables coming from different databases that could be potentially of different types (for example one collection provides access to a PostgreSQL table and another to a MongoDB collection) and in local and/or remote servers. JSON is used as input and output data format.

Once collections are imported or created from scratch, the gLibrary REST API can be used to retrieve, create, update and delete collection's records, that in gLibrary lingo are called **items**, and to manage **replicas**. A powerful filtering system is available to make queries on collections. All criteria are specified using the query string of the REST API call. (e.g., /v1/repos/fantasy\_company/orders?filter[where][userId]=acaland&filter[where][orderQuantity][gt]=200&filter[where][limit]=100 will search for 100 orders issued by the user “acaland” with a quantity of 100).

Each item can have one or more *attachments*, which are called **replica**. Replicas can be stored on Grid Storage Elements (using the Disk Pool Manager – DPM - functionalities) and/or on Cloud Storage (OpenStack Swift is currently supported). The service offers API calls for uploading and downloading files to the configured storages, using a system based on Temporary URLs that permits direct data transfers from clients to destination storages and vice versa, without caching them on the gLibrary intermediate server.

**Relations** between two collections of the same repository can be created, if foreign keys are properly assigned. Currently we support one-to-many relations.

gLibrary can be useful for users that need a secure way to save and share their assets. REST API calls are authenticated using a session token based system, while authorization to repositories, collections and items can be selectively changed based on Access Control Lists (ACLs) and Roles.

Being a API based service, gLibrary can be easily integrated with other services, in particular web applications, Science Gateways, command line jobs, desktop and mobile applications.

For the previous version of gLibrary (1.0), several front-ends were available: a standalone web application[[5]](#footnote-5) and several portlets deployed in a number of Liferay-based Science Gateways that offer feature to browse various repositories, download replicas, upload and register files to Grid and Cloud storages.

In the context of the DARIAH CC project, the new gLibrary 2.0 API are used to offer access to datasets provided by Austrian Academy of Sciences from Science Gateways and mobile applications.

In the context of the DARIAH CC project, the Node.JS API Framework LoopBack[[6]](#footnote-6) has been used to extend the architecture with new functionalities to import datasets already available on different backends.

## gLibrary architecture with Science Gateways

gLibrary 2.0 has been rewritten from scratch and based on the Node.JS API Framework LoopBack. The high-level architecture of the gLibrary 2.0 deployed with a Science gateway or mobile device front-end is sketched in Fig.1



***Figure 1 – The gLibrary architecture in combination with SGs***

When used in combination with Science Gateways and federated authentication, an API gateway (generally deployed in the Science Gateway machine) is needed:

* The AuthN & AuthZ module: gLibrary currently supports the Federated Authentications, the traditional PKI authentications based on the adoption of X.509 certificates and based on session tokens in exchange for a username/password couple. The access to the Grid and Cloud storages are secured using the digital credentials generated on-the-fly by the eTokenServer, the standard-based solution developed by the INFN for central management of robot certificates and provisioning of proxies to get seamless and secure access to computing e-Infrastructures, based on Grid and Cloud middleware supporting X.509 standard for authorization;
* As mentioned in the previous section, several backends can be used to store metadata and data records and for files (local storage systems, Grid Storage Elements based on DPM[[7]](#footnote-7) services with WebDAV interface and Cloud Storage (OpenStack Swift is supported). In particular, the exploitation of the EGI Federated Cloud Infrastructure will offer to the A&H community a distributed storage infrastructure accessible in a seamless and transparent way. This will contribute to increase the resilience and the availability of the solution provided to this community.

## The Semantic Search Engine High-Level Service architecture

Essential improvements for the A&H research community are to deal with RDF, XML files and SPARQL queries on different Open Access Document Repositories (OADRs), semantically enrich OADRs and Data Repositories (DRs) with a specific ontology, and build a search engine on the related linked data to discover new science and support the reproducibility of science.

With this regard, INFN provided access to the Parallel Semantic Search Engine (PSSE) framework that has been customized to address the requirements of the DARIAH CC project. Thanks to this framework, users from the A&H research community can search in the e-Infrastructure Knowledge Base, in more than 100 languages across more than 30 million resources contained in the thousands of semantically enriched OADRs and DRs.

The multi-layered architecture of the Semantic Search Engine is sketched in Fig. 2 where both the official and “de facto” Semantic Web standards and technologies[[8]](#footnote-8) adopted are described by small logos. Starting from the bottom of Fig. 2, the first two components of the service are described below.



***Figure 2 – Architecture of the Semantic Search Engine***

### Metadata Harvester

As shown in Fig. 2, the metadata harvester is a process running either on a Grid or on a Cloud infrastructure which consists of the following parts:

* Get the address of each repository publishing an OAI-PMH standard[[9]](#footnote-9) endpoint;
* Retrieve, using the OAI-PMH repository address, the related Dublin Core[[10]](#footnote-10) encoded metadata in XML format;
* Get the records from the XML files and, using the Apache Jena API[[11]](#footnote-11), transform the metadata in RDF format;
* Save the RDF files into a Virtuoso[[12]](#footnote-12) triple store according to an OWL-compliant ontology built using Protègè[[13]](#footnote-13).

### Semantic-Web Enricher

Each RDF file retrieved and saved in the Virtuoso triple store is mapped onto a Virtuoso Graph that contains the ontology expressly deployed for the search engine, shown in Fig. 3 for the sake of completeness. The ontology, built using Dublin Core and FOAF standards, consist of:

* Classes that describe the general concepts of the domain: Resource, Author, Organisation, Repository and Dataset (where Resource is a given open access document);
* Object properties that describe the relationship among the ontology classes;
* Data properties (or attributes) that contain the characteristics or classes’ parameters.

The third, and high-level, component is the Search Engine itself, which is described in details in the next sub-section.



***Figure 3 – Schema of the ontology used for the Semantic Search Engine***

### Search Engine

The home page of the Parallel Search Engine customized for the DARIAH CC project is shown in Fig. 4. Using the portlet, visitors can enter a keyword (or more keyword at the same time) and submit a SPARQL query to the Virtuoso triple store. The framework is currently configured to simultaneously search across the above e-Infrastructure Knowledge Base[[14]](#footnote-14), Europeana[[15]](#footnote-15), Cultura Italia[[16]](#footnote-16), Isidore[[17]](#footnote-17), OpenAgris[[18]](#footnote-18), PubMed[[19]](#footnote-19) and DBpedia[[20]](#footnote-20) platforms. Others repositories can easily be added since the framework has been designed with modular plugins and the searches are made in parallel using different Java threads.





 ***Figure 4 – Typical results of the given query***

The results of a given query are listed in the summary view shown at the bottom part of Fig. 4. For each record found, the title, the author(s) and a short description of the corresponding resource are provided. Clicking on “More Info”, visitors can access the detailed view of the resource, as shown in Fig. 5.



*Figure 5 – Detailed view of a record found by the Semantic Search Engine*

In the “Dataset information” panel users get the link to the open access document and, if existing, to the corresponding dataset. Clicking on the “Altmetrics”[[21]](#footnote-21) link, it is possible to monitor the impact of the discovered result in the academia. Clicking on the “Linked data” tab, which appears at the top of the summary view, users can select one of more of the resources found and get a graphic view of the semantic connections among Authors, Subjects and Publishers, as shown in Fig. 6.



*Figure 6 – Representation of the resource with all its metadata with LodLive*

## Integration and dependencies

gLibrary 2.0 requires a **MongoDB** (>3.x)[[22]](#footnote-22) server running. It is used to maintain repositories' configurations and also for metadata and data records storage for local repositories. Node.js (> 0.12)[[23]](#footnote-23) is required too.

Being a server service, a tool that ensures it indefinitely runs or restarts automatically is recommended. We have successfully deployed gLibrary both with forever[[24]](#footnote-24) and pm2[[25]](#footnote-25). For testing purposes, Nodemon[[26]](#footnote-26) is a valid option.

The parallel Semantic Search Engine framework can easily be installed deploying a simple JSR 286 compliant portlet in the Application Server. For this pilot we have tested the parallel Semantic Search Engine on GlassFish (v3.1.1) as Application Server.

To graphically represent the resource with all its metadata with LodLive[[27]](#footnote-27), the tool has to be installed separately in the application server. Instructions to install and configure LodLive are the following:

]$ sh <glassfish\_path>/bin/asadmin –u liferayadmin \
 –W <glassfish\_path>/domain/liferay/config/local-password deploy LodLiveGraph.war

# Requirements

Cloud technologies are mature enough to improve the reliability and the availability of data. Clouds can provide the possibility to dynamic data replication among computing centres in a user-transparent way, preventing data loss due to localized disasters. In this deliverable we have presented a solution to manage digital repository in cloud-based infrastructure.

## Functional Requirements

* The parallel SSE accepts keywords as input. These keywords will be used to run SPARQL queries to the Virtuoso triple-store;
* The parallel SSE can be deployed in the Application server and be accessed as a Web service;
* The harvester process can be executed both on Grid and Cloud infrastructures.

## Non-functional Requirements

**Performance**

* Speed and fault tolerance for the parallel SSE have been improved.

**Availability**

* Source codes for gLibrary and the SSE are available on GitHub;
* Travis Integration tests are implemented.

# Feedback on satisfaction

*Who was involved in testing and what the outcome of the review was …*

# Future plans

To raise the awareness of A&H researchers about the necessity of digital research, the two frameworks will be presented during workshops, webinars and/or training courses organized by the DARIAH CC project. This will contribute to advertise the project results and, based on the feedback received from the community, pave the way to drive the further development of these frameworks and/or identify new collaboration of collaboration.

Future developments include the possibility to:

* Enrich the information accessed through the Semantic Search Engine including some general-purpose and domain-specific repositories (e.g. GENONAMES[[28]](#footnote-28), GERMANET[[29]](#footnote-29) and ISLEX[[30]](#footnote-30));
* Extend the ontology based on Protègè to include DARIAH-specific knowledge;
* Discuss about possible integration with other high-level tools developed in the context of the DARIAH CC project;
* Include some other DARIAH-specific tools.
1. gLibrary 2.0 API Docs

Server endpoint

http://glibrary.ct.infn.it:3500

Authentication

Before to send any request, users should be authenticated. Currently authentication is based on username/password couple. This will return a **session token id** that needs to be used with any following request. There are two options to send the **access\_token**:

* via a query parameter:

 curl -X GET http://glibrary.ct.infn.it:3500/v2/repos?access\_token=6Nb2ti5QEXIoDBS5FQGWIz4poRFiBCMMYJbYXSGHWuulOuy0GTEuGx2VCEVvbpBK

* via HTTP headers:

ACCESS\_TOKEN=6Nb2ti5QEXIoDBS5FQGWIz4poRFiBCMMYJbYXSGHWuulOuy0GTEuGx2VCEVvbpBK

curl -X GET -H "Authorization: $ACCESS\_TOKEN" \
http://glibrary.ct.infn.it:3500/v2/repos

Login

To obtain a session id, you need to pass a valid username and password to the following endpoint:

POST /v2/users/login HTTP/1.1

{
 "username":"admin",
 "password":"opensesame2015"
}

Alternatively you can use the email address instead of the username.

User creation

New users are created issuing requests to the following endpoint:

POST /v2/repos/users HTTP/1.1

The mandatory parameters are:

* **username**
* **email**
* **password**

Please notice that the created user, has no access to any repository yet. The admin user need to assign the created user to any repository and/or collections, setting properly the ACLs.

Authorization

Currently gLibrary allows to set separate permissions to repositories, collections and items per each user. The default permission set to a newly created user is *NO ACCESS* to anything. It's admin's responsibility to set properly the ACLs per each user. Currently an instance of gLibrary server has just one superadmin (the *admin* user), but in future releases you will have the option to define admins per repository.

ACLs

To set ACLs, the super admin can issue requests to two separate endpoints:

POST /v2/repos/<repo\_name>/\_acls http/1.1

and/or

POST /v2/repos/<repo\_name>/<collection\_name>/\_acls http/1.1

The body of each requests has the following attributes:

|  |  |
| --- | --- |
| attribute | Description |
| *username* | the username of the user to which we are adding permissions to |
| *permissions* | valid options are "R" and "RW" |
| *items\_permissions* | (for collections only) valid options are "R" and "RW" |

*permissions* refers to repository or collection permission, according to where the request is issued:

* Repository:
	+ "R" grants a user the capability of listing its content (i.e. list of collections)
	+ "RW" grants a user the capability of creating (or importing) new collections or deleting them
* Collection:
	+ "R" grants a user the capabilities to list the collection's content (list of items)
	+ "RW" grants a user the capabilities of creating, updating, deleting the collection's items

*items\_permissions* is valid only for collections' ACL and refers to:

* "R" grants a user the capability to download items' replicas
* "RW" grants a user the capability to create, update and upload replicas

Repositories

A gLibrary server can host one or more **repositories**. A repository should be created before creating new **collections** or importing existing db tables or NoSQL collections as gLibrary collections.

A repository has a name, a path, that represents the access point in the API path, and optionally a coll\_db (*TODO*: rename as default\_collection\_db). If a default DB is defined at the moment of the creation, this will be the default backend DB for all the collections created or imported of the given repository. However, this can be overridden per each collection, if new DB info is provided when the collection is created

List of all the repositories hosted on the server

GET /v2/repos/ HTTP/1.1

Returns a list of all the repositories managed by the given gLibrary server. Each repository has the following properties:

|  |  |
| --- | --- |
| name | Description |
| name | Repository name |
| path | Direct endpoint of the given repository |
| collection\_db | Default database where collection data should be stored. Can be overriden per collection |
| host | FQDN of the default collection DB |
| port | port number of the default collection DB |
| username | username of the default collection DB |
| password | password of the default collection DB |
| database | name of the database to use for the default collection DB |
| type | type of the default collection db (mysql, postgresql, mongodb) |

Example:

{
 "name": "infn",
 "path": "http://glibrary.ct.infn.it:5000/v2/infn",
 "coll\_db": {
 "host": "giular.trigrid.it",
 "port": 3306,
 "username": "root",
 "password": "\*\*\*\*\*\*\*\*\*\*\*\*\*",
 "database": "test",
 "type": "mysql"
 }
}

Each repository can have a collection\_db where collections data will be stored. If no collection\_db is specified, the repository will use the local non-relational mongoDB that comes with gLibrary. Each repository's collection can override the collection\_db.

Create a new repository

POST /v2/repos/ HTTP/1.1

Create a new repository. A default collection\_db can be specified. It will store all the collections in case no collection\\_db parameter is specified during collection creation. This property is optional. If missing it will use the local MongoDB server.

**Parameters**

|  |  |  |
| --- | --- | --- |
| name | type | Description |
| name | string | Name of the repository (will be the API path) |
| collection\_db | object | (Optional) Default database where collection data should be stored. Can be overridden per collection |
| host | string | FQDN of the default collection DB |
| port | number | port number of the default collection DB |
| username | string | username of the default collection DB |
| password | string | password of the default collection DB |
| database | string | name of the database to use for the default collection DB |
| type | string | type of the default collection db (mysql, postgresql, mongodb) |
| default\_storage | object | (Optional) specifies the default storage for replicas |
| baseURL | string | it's full path of Swift Container or Grid SURL for replica uploads |
| type | string | "swift" or "grid" storage |

Note: name is a lowercase string. Numbers and underscores are allowed. No special characters are permitted

Example:

POST /v2/repos/ HTTP/1.1
Content-Type: application/json

{
 "name": "infn",
 "collection\_db": {
 "host": "glibrary.ct.infn.it",
 "port": 5432,
 "username": "infn\_admin",
 "password": "\*\*\*\*\*\*",
 "database": "infn\_db",
 "type": "postgresql"
 },
 "default\_storage": {
 "baseURL": "http://stack-server-01.ct.infn.it:8080/v2/AUTH\_51b2f4e508144fa5b0c28f02b1618bfd/gridcore",
 "type": "swift"
 }
}

Be sure to set Content-Type to application/json in the *Request Headers*.

Collections

Each repository contains one or more collections. Collections are abstractions over relational database tables or non-relational database "collections", exposing their records over REST API and JSON format. The available API allow the repository administrator to create new collection, specifying a schema in the case of relational collection, or importing existing tables/NoSQL collections. If not specified, collections will be created/imported from the default coll\_db (*TODO*: default\_collection\_db) of the containing repository. Otherwise, each collection can retrieve data from local or remote database, overriding the default repository value, using the coll\_db (*TODO*: collection\_db) property.

Create a new collection

POST /v2/repos/<repo\_name>/ HTTP/1.1

**Parameters**

|  |  |  |
| --- | --- | --- |
| name | type | description |
| name | string | Name of collection |
| schema | object | (Optional for non-relational DB) define the schema of the new collection |
| collection\_db | string | (Optional) Default database where collection data should be stored. Can be overridden per collection |
| host | string | FQDN of the default collection DB |
| port | number | port number of the default collection DB |
| username | string | username of the default collection DB |
| password | string | password of the default collection DB |
| database | string | name of the database to use for the default collection DB |
| type | string | type of the default collection db (mysql, postgresql, mongodb) |

Schema is a JSON object listing the name of the attributes and their types in case we want a non-relational collection. Each property represents the name of an attribute and the value is another object with the following keys:

|  |  |
| --- | --- |
| name | description |
| type | type of the attribute's value. Example of allowed types are: string, number, 'boolean', 'date' |
| required | whether a value for the property is required |
| default | default value for the property |
| id | whether the property is a unique identifier. Default is false |

For a full list of the supported type, please refer to [https://docs.strongloop.com/display/public/LB/LoopBack+types](https://docs.strongloop.com/display/public/LB/LoopBack%2Btypes) and [https://docs.strongloop.com/display/public/LB/Model+definition+JSON+file#ModeldefinitionJSONfile-Generalpropertyproperties](https://docs.strongloop.com/display/public/LB/Model%2Bdefinition%2BJSON%2Bfile#ModeldefinitionJSONfile-Generalpropertyproperties).

**Example** (creation of a new collection on a relational db):

POST /v2/repos/infn/ HTTP/1.1
Content-Type: application/json

{
 "name": "articles",
 "schema": {
 "title": {"type": "string", "required": true},
 "year": "integer",
 "authors": "array"
 }
}

The previous request will create a collection named articles into the infn repository. The collection data will be stored into the default coll\_db specified for the infn repository (that according to the previous example is a postgreSQL db named infn\_db)

Import data from an existing relational database

If you want to create a collection that maps an existing db table, two additional properties are available:

|  |  |
| --- | --- |
| name | description |
| import | it should set to true |
| tablename | name of the database table of the database to be imported |

**Example** (creation of a new collection with data coming from an existing relational db):

POST /v2/repos/infn/ HTTP/1.1
Content-Type: application/json

{
 "name": "old\_articles",
 "import": "true",
 "tablename": "pubs",
 "collection\_db": {
 "host": "somehost.ct.infn.it",
 "port": 3306,
 "username": "dbadmin",
 "password": "\*\*\*\*\*\*",
 "database": "test\_daily",
 "type": "mysql"
 }}

The previous request will create the collection old\_articles import data from an existing database, named test\_daily and providing access to its table named pubs.

List all the collections of a repository

GET /v2/repos/<repo\_name>/ HTTP/1.1

This API will return a JSON array with all the collections of <repo\_name>. Each collection will have a schema attribute, describing the schema of the underlying DB table. If the schema attribute is null it means the collection has been imported and it inherits the schema of the underlying DB table. An additional API is available to retrieve the schema of a given collection (see [next API](#retrieve-the-schema-of-a-collection)).

**Example**

GET /v2/repos/sports HTTP/1.1

[
 {
 "id": "560a60987ddaee89366556d2",
 "name": "football",
 "path": "/sports/football",
 "location": "football",
 "coll\_db": null,
 "import": "false",
 "schema": null
 },
 {
 "id": "560a60987ddaee89366556d3",
 "name": "windsurf",
 "path": "/sports/windsurf",
 "location": "windsurf",
 "coll\_db": null,
 "import": "false",
 "schema": {
 "rider": {
 "type": "string",
 "required": true
 },
 "nationality": {
 "type": "string",
 "required": false
 },
 "teamid": {
 "type": "number",
 "required": false
 }
 }
 }
]

The sports repository has two collections football and windsurf. The first one is stored on the default coll\_db repository DB and it's schema-less, while the second one has a predefined schema.

Retrieve the schema of a collection

GET /v2/repos/<repo\_name>/<collection\_name>/\_schema HTTP/1.1

If the given collection\_name is hosted in a relation database table, this API will return a JSON object with the schema of the underlying table.

**Example**

GET /v2/repos/comics/dylandog/\_schema HTTP/1.1

{
 "id": {
 "required": true,
 "length": null,
 "precision": 10,
 "scale": 0,
 "id": 1,
 "mysql": {
 "columnName": "id",
 "dataType": "int",
 "dataLength": null,
 "dataPrecision": 10,
 "dataScale": 0,
 "nullable": "N"
 }
 },
 "fragebogenId": {
 "required": true,
 "length": null,
 "precision": 10,
 "scale": 0,
 "mysql": {
 "columnName": "fragebogen\_id",
 "dataType": "int",
 "dataLength": null,
 "dataPrecision": 10,
 "dataScale": 0,
 "nullable": "N"
 }
 },
 "nummer": {
 "required": true,
 "length": 256,
 "precision": null,
 "scale": null,
 "mysql": {
 "columnName": "nummer",
 "dataType": "varchar",
 "dataLength": 256,
 "dataPrecision": null,
 "dataScale": null,
 "nullable": "N"
 }
 }
}

Delete a collection

DELETE /v2/repos/<repo\_name>/<collection\_name> HTTP/1.1

This API will delete the given collection\_name from repo\_name. Actual data on the backend table should not be deleted. It's a sort of *unlinking*, so that the db table/nosql collection will not be accessible anymore from the gLibrary REST API.

Items (previously entries)

**Items** represents the content of a given collection. If a collection is hosted in a relational database, each item is a table record, while if it's non-relational it's the document/object of the NoSQL collection. Items can be listed and queried via the filtering system, created/added, updated and deleted, using the REST API provided by gLibrary.

Item creation

POST /v2/repos/<repo\_name>/<collection\_name> HTTP/1.1

This API add a new item into the given collection\_name. Item content have to be provided as a JSON object. In case of the relational collection it should conform to the collection schema. In the case of attributes that have no corresponding column table, their values will be ignored silently. If the API will be successful a new record or document will be added to the underlying table or NoSQL collection.

**Example**

POST /v2/repos/infn/articles HTTP/1.1

{
 "title": "e-Infrastructures for Cultural Heritage Applications",
 "year": 2010,
 "authors": [ "A. Calanducci", "G. Foti", "R. Barbera" ]
}

Item listing

GET /v2/repos/<repo\_name>/<collection\_name>/ HTTP/1.1

Retrieve the items inside the collection\_name as a JSON array of objects. Each object is a record of the underlying table (in case of relational DB) or document (in case of NoSQL collection). By default the first 50 items are returned. See below the description of filtering system in the [query section](#queries-with-filters) to change this behavior.

**Example**

GET /v2/repos/gridcore/tracciati HTTP/1.1

Item detail

GET /v2/repos/<repo\_name>/<collection\_name>/<item\_id> HTTP/1.1

Retrieve the detail of an item with a given\_id. It will return a JSON object with the attributes mapping the schema of the given collection\_name.

**Example**

GET /v2/repos/infn/articles/22

Item deletion

DELETE /v2/repos/<repo\_name>/<collection\_name>/<item\_id> HTTP/1.1

Delete the given item\_id of the the collection collection\_name. Delete will be successful only if the given item has no replica. You can force the deletion of item with replicas setting:

{
 "force": true
}

in the request body.

Item update

PUT /v2/repos/<repo\_name>/<collection\_name>/<item\_id> HTTP/1.1

Update one of more attributes of the given item\_id. The request body has to contain a JSON object with the attribute-value pair to be updated with the new values.

Queries with filters

GET /v2/repos/<repo\_name>/<collection\_name>?filter[<filterType>]=<spec>&filter[...]=<spec>... HTTP/1.1

where filterType is one of the following:

* where
* include
* order
* limit
* skip
* fields

and spec is the specification of the used filter.

Additional info on the full query syntax can be found [here](https://docs.strongloop.com/display/public/LB/Querying%2Bdata#Queryingdata-RESTsyntax)

Replicas

Each item can have one or more attachments, generally the same file stored in different locations, such as Cloud storage servers (Swift based) or Grid Storage Elements (DPM based). So we call them also replicas.

Replica creation

POST /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/\_replicas/ HTTP/1.1

|  |  |
| --- | --- |
| name | description |
| uri | (optional) provides the full storage path of where the replica will be saved |
| type | (optional) specifies the type of storage backend. Currently "swift" or "grid" |
| filename | The filename of the given replica |

The first two parameters (uri and type) are optionals if a default\_storage attribute has been set for the given repository. If not, they need to be specified, otherwise the request to the API will fail.

Please note that this API will just create a replica entry for the item, but no actual file will be uploaded from the client. Once the replica has been created you need to use the **Upload** API to transfer the actual file payload.

Retrieve all the replicas of the given item\_id

GET /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/\_replicas/ HTTP/1.1

Download a given replica

GET /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/\_replicas/<rep\_id> HTTP/1.1

Upload a replica

Upload the file payload to the destination storage. This requires two subsequent API request.

First, ask for the destination endpoint for the upload with:

PUT /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/\_replicas/<rep\_id> HTTP/1.1

This will return a **temporaryURL** valid a few seconds (example):

{
 "uploadURI": "http://stack-server-01.ct.infn.it:8080/v2/AUTH\_51b2f4e508144fa5b0c28f02b1618bfd/gridcore/ananas.jpg?temp\_url\_sig=6cd7dbdc2f9e429a1b89689dc4e77f1d2aadbfc8&temp\_url\_expires=1449481594"
}

Then use the URL returned by the previous API to upload the actual file, using the PUT verb again (example):

PUT http://stack-server-01.ct.infn.it:8080/v2/AUTH\_51b2f4e508144fa5b0c28f02b1618bfd/gridcore/ananas.jpg?temp\_url\_sig=6cd7dbdc2f9e429a1b89689dc4e77f1d2aadbfc8&temp\_url\_expires=1449481594 HTTP/1.1

It will return a 201 status code, if the upload will complete successfully

Delete a replica

DELETE /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/\_replicas/<rep\_id> HTTP/1.1

Relations

One to many relations can be created between collections of the same repository, properly setting a foreign key.

To set the relation among two collections, issue the following request to the collection in the "one" side of the one-to-many relation:

POST /v2/repos/<repo\_name>/<collection\_name>/\_relation HTTP/1.1

The body of the request needs to provide two attributes:

|  |  |
| --- | --- |
| name | description |
| *relatedCollection* | the "many" side of the one-to-many relation |
| *fk* | the *foreign key* of *relatedCollection* that match the *id* of <*collection\_name*> |

In practice, you should set the *fk* in such a way collection\_name.id == relatedCollection.fk

Retrieve related items

GET /v2/repos/<repo\_name>/<collection\_name>/<item\_id>/<related\_collection>

Retrieve all the items from related\_collection of the given item\_id.

Contributors

Antonio Calanducci (antonio.calanducci@ct.infn.it)

Antonio Di Mariano (antonio.dimariano@gmail.com)

1. <https://wiki.egi.eu/wiki/Competence_centre_DARIAH> [↑](#footnote-ref-1)
2. <https://www.egi.eu/infrastructure/cloud/> [↑](#footnote-ref-2)
3. [www.oeaw.ac.at](http://www.oeaw.ac.at) [↑](#footnote-ref-3)
4. <http://repositories.webometrics.info/> [↑](#footnote-ref-4)
5. <https://glibrary.ct.infn.it> [↑](#footnote-ref-5)
6. <http://loopback.io/> [↑](#footnote-ref-6)
7. [www.gridpp.ac.uk/wiki/DPM\_Install](http://www.gridpp.ac.uk/wiki/DPM_Install) [↑](#footnote-ref-7)
8. <http://semanticweb.org/wiki/Semantic_Web_standards> [↑](#footnote-ref-8)
9. [www.openarchives.org/phm](http://www.openarchives.org/phm) [↑](#footnote-ref-9)
10. [www.dublincore.org](http://www.dublincore.org) [↑](#footnote-ref-10)
11. <http://jena.apache.org> [↑](#footnote-ref-11)
12. <http://virtuoso.openlinksw.com> [↑](#footnote-ref-12)
13. <http://protege.standford.edu> [↑](#footnote-ref-13)
14. [www.chain-reds.eu/web/old-project/knowledge-base](http://www.chain-reds.eu/web/old-project/knowledge-base) [↑](#footnote-ref-14)
15. <http://europeana.ontotext.com/sparql> [↑](#footnote-ref-15)
16. <http://dati.culturaitalia.it/sparql> [↑](#footnote-ref-16)
17. <http://rechercheisidore.fr/sparql/> [↑](#footnote-ref-17)
18. [http://202.45.139.84:10035/catalogs/fao/repositories/agris#query](http://202.45.139.84:10035/catalogs/fao/repositories/agris%22%20%5Cl%20%22query) [↑](#footnote-ref-18)
19. <http://pubmed.bio2rdf.org/sparql> [↑](#footnote-ref-19)
20. <http://dbpedia.org/sparql> [↑](#footnote-ref-20)
21. <http://www.altmetric.com/> [↑](#footnote-ref-21)
22. <https://www.mongodb.org> [↑](#footnote-ref-22)
23. <https://nodejs.org/> [↑](#footnote-ref-23)
24. <https://github.com/foreverjs/forever> [↑](#footnote-ref-24)
25. <https://github.com/Unitech/pm2> [↑](#footnote-ref-25)
26. <https://github.com/remy/nodemon> [↑](#footnote-ref-26)
27. <https://github.com/dvcama/LodLive> [↑](#footnote-ref-27)
28. [www.genonames](http://www.genonames) [↑](#footnote-ref-28)
29. [www.sfs.uni-tuebingen.de/GermaNet/](http://www.sfs.uni-tuebingen.de/GermaNet/) [↑](#footnote-ref-29)
30. [www.islex.is/](http://www.islex.is/) [↑](#footnote-ref-30)