KurentoRepository Documentation

Release 6.6.1-dev

kurento.org

Mar 27, 2020

Contents

1 Introduction **3**
- 1.1 Example . . . 4
- 1.2 Source code . . . 4
- 1.3 Content of this documentation . . . 4

2 Repository Server **5**
- 2.1 Dependencies . . . 5
- 2.2 Binaries . . . 5
- 2.3 Configuration . . . 6
- 2.4 Logging configuration . . . 6
- 2.5 Execution . . . 7
- 2.6 Run at user-level . . . 7
- 2.7 Run as daemon . . . 7
- 2.8 Version upgrade . . . 7
- 2.9 Installation over MongoDB . . . 7

3 Repository REST API **9**
- 3.1 Create repository item . . . 9
- 3.2 Remove repository item . . . 10
- 3.3 Get repository item read endpoint . . . 11
- 3.4 Find repository items by metadata . . . 11
- 3.5 Find repository items by metadata regex . . . 12
- 3.6 Get the metadata of a repository item . . . 13
- 3.7 Update the metadata of a repository item . . . 14

4 Repository Client **15**
- 4.1 Repository Java API . . . 15
- 4.2 Repository internals . . . 15

5 Glossary **17**

Index **19**


KURENTO

CHAPTER 1

Introduction

Welcome to the **Kurento Repository** project. This piece of software is part of Kurento, and it allows to manage media repositories based on the storage capabilities provided by a file system or an instance of *MongoDB*. The following picture illustrates the architecture of the Kurento Repository:

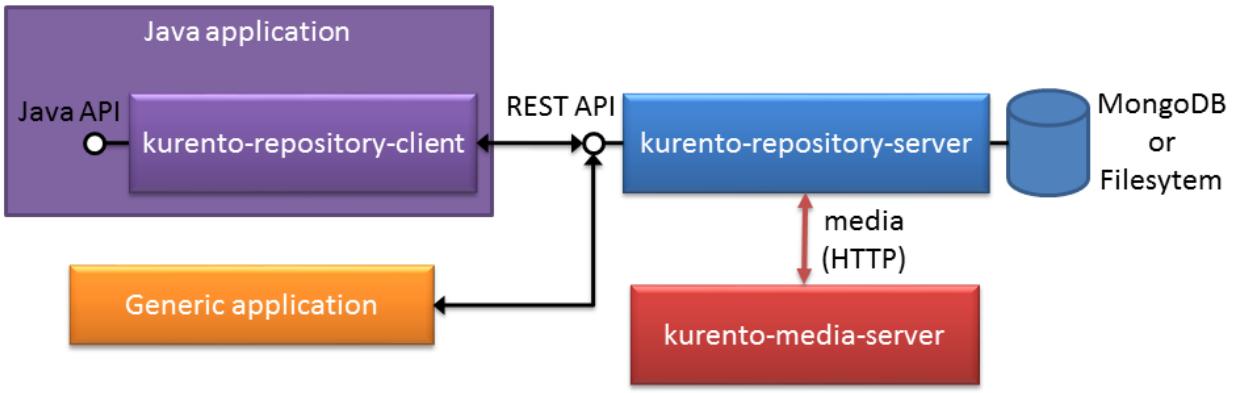

Fig. 1: *Kurento Repository Architecture*

As it can be seen, the Kurento Repository has been designed following a client-server architecture. Therefore, it has two main components:

- **kurento-repository-server** : Stand-alone application which implements a media repository, exposing its capabilities through an easy-to-use *HTTP REST* API. It has been implement as a *Spring Boot* application.
- **kurento-repository-client** : Java library that wraps the REST communication with the server, exposing a Java API which can be consumed in Java applications.

In short, the *kurento-repository-server* allows media recording and playing. These capabilities can be consumed by any application by means of the REST API, of specifically by Java application using the *kurento-repository-client*. The media in the repository can be consumed by *kurento-media-server* using HTTP as transport.

1.1 Example

There is a Kurento Java tutorial application that connects to a running instance of the **kurento-repository-server** to record and play media over HTTP using the capabilities of the Kurento Media Server.

1.2 Source code

Kurento Repository is open source (Apache 2.0) and it is hosted on GitHub. This Git repository contains a Maven project with the following modules:

- *kurento-java*: Reactor project.
- *kurento-repository-server*: Contains a Spring Boot application, very easy to setup and run. It exposes the HTTP REST API
- *kurento-repository-client*: Wrapper for the library, offering a simpler Java API.
- *kurento-repository-internal*: core Java library of repository. Plain Java mostly, but with Spring dependencies.

1.3 Content of this documentation

In order to learn how to build, configure and run the server application, please refer to the *Repository Server* section. The *REST API* that can be used to communicate with the server are described in the next section. The Java API is depicted on the section *repository client*. The complete JavaDoc reference is also provided. Finally, a reference of the most important terms handled in this documentation is summarized in the *glossary* section.

CHAPTER 2

Repository Server

This section explains how to build configure and run the Kurento Repository server as a standalone application.

2.1 Dependencies

- Ubuntu 14.04 LTS
- Java JDK version 7 or 8
- *MongoDB* (we provide an *install guide*)
- *Kurento Media Server* or connection with a running instance (to install follow the official guide)

2.2 Binaries

To build the installation binaries from the source code you'll need to have installed on your machine Git, Java JDK and *Maven*.

Clone the parent project, **kurento-java** from its GitHub repository.

```
$ git clone git@github.com:Kurento/kurento-java.git
```

Then build the **kurento-repository-server** project together with its required modules:

```
$ cd kurento-java
$ mvn clean package -DskipTests -Pdefault -am \
   -pl kurento-repository/kurento-repository-server
```

Now unzip the generated install binaries (where `x.y.z` is the current version and could include the `-SNAPSHOT` suffix):

```
$ cd kurento-repository/kurento-repository-server/target
$ unzip kurento-repository-server-x.y.z.zip
```

2.3 Configuration

The configuration file, `kurento-repo.conf.json` is located in the `config` folder inside the uncompressed installation binaries. When installing the repository as a system service, the configuration files will be located after the installation inside `/etc/kurento`.

```
$ cd kurento-repository-server-x.y.z
$ vim config/kurento-repo.conf.json
```

The default contents of the configuration file:

```
{
  "repository": {
    "port": 7676,
    "hostname": "127.0.0.1",

    //mongodb or filesystem
    "type": "mongodb",

    "mongodb": {
      "dbName": "kurento",
      "gridName": "kfs",
      "urlConn": "mongodb://localhost"
    },
    "filesystem": {
      "folder": "/tmp/repository"
    }
  }
}
```

These properties and their values will configure the repository application.

- `port` and `hostname` are where the HTTP repository servlet will be listening for incoming connections (REST API).
- `type` indicates the storage type. The repository that stores media served by KMS can be backed by GridFS on MongoDB or it can use file storage directly on the system's disks (regular filesystem).
- **`mongodb` configuration:**
  - `dbname` is the database name
  - `gridName` is the name of the gridfs collection used for the repository
  - `urlConn` is the connection to the Mongo database
- **`filesystem` configuration:**
  - `folder` is a local path to be used as media storage

2.4 Logging configuration

The logging configuration is specified by the file `kurento-repo-log4j.properties`, also found in the `config` folder.

```
$ cd kurento-repository-server-x.y.z
$ vim config/kurento-repo-log4j.properties
```

In it, the location of the server's output log file can be set up, the default location will be `kurento-repository-server-x.y.z/logs/` (or `/var/log/kurento/` for system-wide installations).

To change it, replace the `${kurento-repo.log.file}` variable for an absolute path on your system:

2.5 Execution

There are two options for running the server:

- user-level execution - doesn't need additional installation steps, can be done right after uncompressing the installer
- system-level execution - requires installation of the repository application as a system service, which enables automatic startup after system reboots

In both cases, as the application uses the *Spring Boot* framework, it executes inside an embedded Tomcat container instance, so there's no need for extra deployment actions (like using a third-party servlet container). If required, the project's build configuration could be modified in order to generate a *WAR* instead of a *JAR*.

2.6 Run at user-level

After having *configured* the server instance just execute the start script:

```
$ cd kurento-repository-server-x.y.z
$ ./bin/start.sh
```

2.7 Run as daemon

First install the repository after having built and uncompressed the generating binaries. **sudo** privileges are required to install it as a service:

```
$ cd kurento-repository-server-x.y.z
$ sudo ./bin/install.sh
```

The service **kurento-repo** will be automatically started.

Now, you can configure the repository as stated in the *previous section* and restart the service.

```
$ sudo service kurento-repo {start|stop|status|restart|reload}
```

2.8 Version upgrade

To update to a newer version, it suffices to follow once again the installation procedures.

2.9 Installation over MongoDB

For the sake of testing *kurento-repository* on Ubuntu (*14.04 LTS 64 bits*), the default installation of MongoDB is enough. Execute the following commands (taken from MongoDB webpage):

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
$ echo "deb http://repo.mongodb.org/apt/ubuntu \
   "$(lsb_release -sc)"/mongodb-org/3.0 multiverse" \
   | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

CHAPTER 3

Repository REST API

This section details the REST API through which an application can communicate with the Kurento Repository Server.

There's also the possibility to integrate the server as a Spring component and in this case the class `org.kurento.repository.RepositoryService` can be used to control the Repository as an instance local to the application. The REST API maps over the service's one, so the methods and parameters involved are the exact same ones.

Primitives provided by the repository server, can be used to control items from the repository (*add*, *delete*, *search*, *update*, *get download URL*).

- *Create repository item*
- *Remove repository item*
- *Get repository item read endpoint*
- *Find repository items by metadata*
- *Find repository items by metadata regex*
- *Get the metadata of a repository item*
- *Update the metadata of a repository item*

3.1 Create repository item

- **Description**: Creates a new repository item with the provided metadata and its associated recorder endpoint.
- **Request method and URL**: `POST /repo/item`
- **Request Content-Type**: `application/json`
- **Request parameters**: Pairs of key-value Strings in JSON format (a representation of the Java object `Map<String, String>`).

Parameter	Type	Description
`keyN`	**O**	Metadata associated to `keyN`
M=Mandatory, O=Optional		

- **Request parameters example**:

```
{
    "key1": "value1",
    "keyN": "valueN"
}
```

- **Response elements**: Returns an entity of type `application/json` including a POJO of type `RepositoryItemRecorder` with the following information:

Element	Type	Description
`id`	**M**	Public ID of the newly created item
`url`	**M**	URL of the item's recording Http endpoint
M=Mandatory, O=Optional		

- **Response example**:

```
{
    "id": "Item's public ID",
    "url": "Recorder Http endpoint"
}
```

- **Response Codes**:

Code	Description
200 OK	New item created and ready for recording.

3.2 Remove repository item

- **Description**: Removes the repository item associated to the provided id.
- **Request method and URL**: `DELETE /repo/item/{itemId}`
- **Request Content-Type**: `NONE`
- **Request parameters**: The item's ID is coded in the URL's path info.

Parameter	Type	Description
`itemId`	**M**	Repository item's identifier
M=Mandatory, O=Optional		

- **Response elements**: `NONE`
- **Response Codes**:

Code	Description
200 OK	Item successfully deleted.
404 Not Found	Item does not exist.

3.3 Get repository item read endpoint

- **Description**: Obtains a new endpoint for reading (playing *multimedia*) from the repository item.
- **Request method and URL**: `GET /repo/item/{itemId}`
- **Request Content-Type**: `NONE`
- **Request parameters**: The item's ID is coded in the URL's path info.

Parameter	Type	Description
`itemId`	**M**	Repository item's identifier
M=Mandatory, O=Optional		

- **Response elements**: Returns an entity of type `application/json` including a POJO of type `RepositoryItemPlayer` with the following information:

Element	Type	Description
`id`	**M**	Public ID of the newly created item
`url`	**M**	URL of the item's reading (playing) Http endpoint
M=Mandatory, O=Optional		

- **Response example**:

```
{
    "id": "Item's public ID",
    "url": "Player Http endpoint"
}
```

- **Response Codes**:

Code	Description
200 OK	New player item created.
404 Not Found	Item does not exist.

3.4 Find repository items by metadata

- **Description**: Searches for repository items by each pair of attributes and their exact values.
- **Request method and URL**: `POST /repo/item/find`
- **Request Content-Type**: `application/json`
- **Request parameters**: Pairs of key-value Strings in JSON format (a representation of the Java object `Map<String, String>`).

Parameter	Type	Description
`searchKeyN`	**M**	Metadata associated to `searchKeyN`
M=Mandatory, O=Optional		

- **Request parameters example**:

```
{
    "searchKey1": "searchValue1",
    "searchKeyN": "searchValueN"
}
```

- **Response elements**: Returns an entity of type `application/json` including a POJO of type `Set<String>` with the following information:

Element	Type	Description
`idN`	**O**	Id of the N-th repository item whose metadata matches one of the search terms
M=Mandatory, O=Optional		

- **Response example**:

```
[ "id1", "idN" ]
```

- **Response Codes**:

Code	Description
200 OK	Query successfully executed.

3.5 Find repository items by metadata regex

- **Description**: Searches for repository items by each pair of attributes and their values which can represent a regular expression (Perl compatible regular expressions).
- **Request method and URL**: `POST /repo/item/find/regex`
- **Request Content-Type**: `application/json`
- **Request parameters**: Pairs of key-value Strings in JSON format (a representation of the Java object `Map<String, String>`).

Parameter	Type	Description
`searchKeyN`	**M**	Regex for metadata associated to `searchKeyN`
M=Mandatory, O=Optional		

- **Request parameters example**:

```
{
    "searchKey1": "searchRegex1",
    "searchKeyN": "searchRegexN"
}
```

- **Response elements**: Returns an entity of type `application/json` including a POJO of type `Set<String>` with the following information:

Element	Type	Description
`idN`	**O**	Id of the N-th repository item whose metadata matches one of the search terms
M=Mandatory, O=Optional		

- **Response example**:

```
[ "id1", "idN" ]
```

- **Response Codes**:

Code	Description
200 OK	Query successfully executed.

3.6 Get the metadata of a repository item

- **Description**: Returns the metadata from a repository item.
- **Request method and URL**: `GET /repo/item/{itemId}/metadata`
- **Request Content-Type**: `NONE`
- **Request parameters**: The item's ID is coded in the URL's path info.

Parameter	Type	Description
`itemId`	**M**	Repository item's identifier
M=Mandatory, O=Optional		

- **Response elements**: Returns an entity of type `application/json` including a POJO of type `Map<String, String>` with the following information:

Element	Type	Description
`keyN`	**O**	Metadata associated to `keyN`
M=Mandatory, O=Optional		

- **Response example**:

```
{
    "key1": "value1",
    "keyN": "valueN"
}
```

- **Response Codes**:

Code	Description
200 OK	Query successfully executed.
404 Not Found	Item does not exist.

3.7 Update the metadata of a repository item

- **Description**: Replaces the metadata of a repository item with the provided values from the request's body.
- **Request method and URL**: `PUT /repo/item/{itemId}/metadata`
- **Request Content-Type**: `application/json`
- **Request parameters**: The item's ID is coded in the URL's path info and the request's body contains key-value Strings in JSON format (a representation of the Java object `Map<String, String>`).

Parameter	Type	Description
`itemId`	**M**	Repository item's identifier
`keyN`	**O**	Metadata associated to `keyN`
M=Mandatory, O=Optional		

- **Request parameters example**:

```
{
    "key1": "value1",
    "keyN": "valueN"
}
```

- **Response elements**: `NONE`
- **Response Codes**:

Code	Description
200 OK	Item successfully updated.
404 Not Found	Item does not exist.

CHAPTER 4

Repository Client

This Java library belonging to the Kurento Repository stack has been designed using the *Retrofit* framework in order to provide a Java wrapper for the *HTTP REST interface* exposed by the repository server.

4.1 Repository Java API

This API maps directly over the *REST interface layer*, in such a way that the primitives exposed by this library mirror the REST ones.

The Java documentation included in `org.kurento.repository.RepositoryClient` and its factory `org.kurento.repository.RepositoryClientProvider` is quite detailed so it shouldn't be very difficult to set up a client connected to a running Kurento Repository Server (don't forget to check our tutorials, and especially the kurento-hello-world-repository).

This library can be imported as a Maven dependency and then instances of `org.kurento.repository.RepositoryClient` can be created in order to interact with the repository server. The Maven artifact to be included in you pom.xml is

```
<dependency>
    <groupId>org.kurento</groupId>
    <artifactId>kurento-repository-client</artifactId>
    <version>6.6.1-SNAPSHOT</version>
</dependency>
```

We provide a Kurento Java tutorial, kurento-hello-world-repository , which uses this library to save the streamed media from a web browser using a repository server instance.

4.2 Repository internals

The internal library of Kurento Repository provides an API that can be used to manage media repositories based on filesystem or *MongoDB*.

The chosen transport for the media is the HTTP protocol. The repository API will provide managed URIs which the application or *Kurento Media Server* can use to upload or download media.

This library can be configured and instantiated as a Spring bean. Although, it shouldn't be used directly but through the *repository server* which offers a REST and Java APIs which should suffice for most applications.

CHAPTER 5

Glossary

This is a glossary of terms that often appear in discussion about multimedia transmissions. Most of the terms are described and linked to its wikipedia, RFC or W3C relevant documents. Some of the terms are specific to *kurento*.

BSON It's a computer data interchange format used mainly as a data storage and network transfer format in the MongoDB database. It is a binary form for representing simple data structures and associative arrays (called objects or documents in MongoDB).

HTTP The is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

See also:

RFC 2616

JSON JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is designed to be easy to understand and write for humans and easy to parse for machines.

JSON-RPC JSON-RPC is a simple remote procedure call protocol encoded in JSON. JSON-RPC allows for notifications and for multiple calls to be sent to the server which may be answered out of order.

Kurento Kurento is a platform for the development of multimedia enabled applications. Kurento is the Esperanto term for the English word 'stream'. We chose this name because we believe the Esperanto principles are inspiring for what the multimedia community needs: simplicity, openness and universality. Kurento is open source, released under Apache 2.0, and has several components, providing solutions to most multimedia common services requirements. Those components include: *Kurento Media Server*, *Kurento API*, *Kurento Protocol*, and *Kurento Client*.

Kurento API **Kurento API** is an object oriented API to create media pipelines to control media. It can be seen as and interface to Kurento Media Server. It can be used from the Kurento Protocol or from Kurento Clients.

Kurento Client A **Kurento Client** is a programming library (Java or JavaScript) used to control **Kurento Media Server** from an application. For example, with this library, any developer can create a web application that uses Kurento Media Server to receive audio and video from the user web browser, process it and send it back again over Internet. Kurento Client exposes the *Kurento API* to app developers.

Kurento Protocol Communication between KMS and clients by means of *JSON-RPC* messages. It is based on *WebSocket* that uses *JSON-RPC* V2.0 messages for making requests and sending responses.

Kurento Media Server **Kurento Media Server** is the core element of Kurento since it responsible for media transmission, processing, loading and recording.

Maven Maven is a build automation tool used primarily for Java projects.

MongoDB MongoDB (from hu**mongo**us) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of *JSON*-like documents with dynamic schemas (MongoDB calls the format *BSON*), making the integration of data in certain types of applications easier and faster. MongoDB is free and open-source software.

See also:

page at Wikipedia

MongoDB official page

Multimedia Multimedia is concerned with the computer controlled integration of text, graphics, video, animation, audio, and any other media where information can be represented, stored, transmitted and processed digitally.

There is a temporal relationship between many forms of media, for instance audio, video and animations. There 2 are forms of problems involved in

- Sequencing within the media, i.e. playing frames in correct order or time frame.
- Synchronisation, i.e. inter-media scheduling. For example, keeping video and audio synchronized or displaying captions or subtitles in the required intervals.

See also:

Wikipedia definition of

REST

is an architectural style consisting of a coordinated set of constraints applied to components, connectors, and data elements, within a distributed hypermedia system. The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

Retrofit Retrofit is a type-safe REST client for Android built by Square. As stated on its webpage, it turns your HTTP API into a Java interface.

See also:

Retrofit home page with documentation and examples

Sphinx Documentation generation system used for Brandtalk documentation.

See also:

Easy and beautiful documentation with Sphinx

Spring Boot Spring Boot is Spring's convention-over-configuration solution for creating stand-alone, production-grade Spring based applications that can you can "just run". It embeds Tomcat or Jetty directly and so there is no need to deploy WAR files in order to run web applications.

WebSocket WebSocket specification (developed as part of the HTML5 initiative) defines a full-duplex single socket connection over which messages can be sent between client and server.

Index

B

BSON, **17**

H

HTTP, **17**

J

JSON, **17**
JSON-RPC, **17**

K

Kurento, **17**
Kurento API, **17**
Kurento Client, **17**
Kurento Media Server, **18**
Kurento Protocol, **17**

M

Maven, **18**
MongoDB, **18**
Multimedia, **18**

R

REST, **18**
Retrofit, **18**
RFC
    RFC 2616, 17

S

Sphinx, **18**
Spring Boot, **18**

W

WebSocket, **18**