# Part 5: Cytomine Java Client

## Cytomine Java Client

### Presentation

Cytomine-java-client is an opensource Cytomine client. It is a JAR file that you can import in your JVM application (Java, Groovy, …).

You will be able to manage data in a Cytomine instance according to your credentials: manage users, add annotations, upload images,...

For more information about Cytomine, go to www.cytomine.be

Requirements:

- Java 1.7+
- Maven 2+ (only if you need to build a new Jar)

Binaries: https://github.com/cytomine/cytomine-java-client/releases

Source code: https://github.com/cytomine/cytomine-java-client

### How to use the Java-client

The java client is available with 2 jars:

- cytomine-java-client-*-jar-with-dependencies: A jar with all classes (client and dependencies).
- cytomine-java-client-*.jar: The jar only contains client classes without dependencies classes. A maven pom.xml is provided.

You need 3 paramters:

- **CYTOMINE_URL**: The full URL of the Cytomine core (“http://...cytomine.be”).
- **PUBLIC_KEY**: Your cytomine public key.
- **PRIVATE_KEY**: Your cytomine private key.

```
Cytomine cytomine = new Cytomine(CYTOMINE_URL, PUBLIC_KEY, PRIVATE_KEY);
```

How to find my key?

Open your browser to your Cytomine instance. Log on and click on your name (right-top), click on Account and go to the “API KEYS” section.

The main class of the client is the Cytomine class. This class provides the big set of methods to retrieve, create, update and delete data in your Cytomine instan

You are now able to perform some actions like:

- add annotations,
- retrieve user with id = xxx
- upload a new image
- list all projects
- ...

Here is a sample code that should print “Hello ” with your Cytomine username and print the list of projects available.

```
import be.cytomine.client.*
import be.cytomine.client.models.*
import be.cytomine.client.collections.*

Cytomine cytomine = new Cytomine(CYTOMINE_URL, PUBLIC_KEY, PRIVATE_KEY);
System.out.println("Hello " + cytomine.getCurrentUser().get("username"));
System.out.println("******* You have access to these projects: *******");
ProjectCollection projects = cytomine.getProjects();
for(int i=0;i<projects.size();i++) {
    Project project = projects.get(i);
    System.out.println(project.get("name"));
}
```

`This can print something like:`

```
Hello johndoe
******* You have access to these projects: *******
PROJECTS_TEST
MY_PROJECTS
...
```

## Plugin Architecture

The plugin architecture is very simple:

- Cytomine.java: The main class, contains a big set of methods to manage data in Cytomine.
- Model.java: This abstract class represents a genereic Cytomine ressource.
- Collection.java: Same as Model.java but for a ressource list.
- All model class go into be.cytomine.client.models package and all collection classes go into be.cytomine.client.collection package.

The main dependencies are:

- HTTPClient library to perform HTTP action.
- JSON-simple library to parse JSON.

### Model.java

Abstract class using a Map. It represents a Cytomine ressource parse from the JSON response.

If you want to add a new ressource, you can create a new class that extend the Model class.

Lets take the example with User ressource (already available in the plugin).

```
class User extends Model {
    String getDomainName() { return "user"}
}
```

The only thing that you need to implement is the getDomainName method. It’s the ressource name from the API URL:

```
/api/user.json
/api/user/$id.json
...
```

=> “user”

The User class will have a Map a full set of getter/setter methods.

```
User user = new User();
user.set("username","johndoe");
String username = user.get("username");
```

Each class extending Model will have a toURL() method. This URL will be used to perform the HTTP action. By default the method content is:

```
String toURL() {
    Integer id = map.get("id")
    if (id) return "/api/" + getDomainName() + ".json"
    else return "/api/" + getDomainName() + "/" + id + ".json"
}
```

In the User case:

```
User user = new User();
user.toUrl() // => /api/user.json
user.set(“id”,”123”)
user.toUrl() // => /api/user/123.json
```

Of course, it's possible to override this method.

## AssociationModel.java

If you have an association ressource (like AnnotationTerm, UserGroup,...), the new class needs to implement AssociationModel (instead of Model). The toURL() method will build standard link thanks to the name of 2 resources.

## Collection.java

Collection is like Model.java, an abstract class but this class provides a List instead of a Map.

```
class UserCollection extends Collection {
    String getDomainName() { return "user"}
    String toURL() {}
    User get(int i) {}
}
```

You can filter the listing with two methods:

- void addFilter(String, String)
- String getFilter(String)

Its toURL() may be implement depending on filters.

For example in UserCollection (Just for example. These URL does not exist in the Cytomine API):

```
String toURL() {
    if (isFilterBy("project")) return "/api/project/" + getFilter("project") + "/" +
getDomainName() + ".json";
    if (isFilterBy("image")) return "/api/" + getDomainName() + ".json?image=" +
getFilter("image");
    else return "/api/" + getDomainName() + ".json";
}
```

```
UserCollection userCollection = new UserCollection();
userCollection.addFilter(“project”,”123”)
userCollection.toURL() // => /api/project/123/user.json

userCollection.addFilter(“image”,”123”)
userCollection.toURL() // => /api/user.json?image=123
//no filter
userCollection.toURL() // => /api/user.json
```

### Collection Pagination

If you have to get a big collection from the server, its better to use pagination (offset/max).
You can set the max item fro the request with:

```
cytomine.setMax(n);
```

And each time you will perform a GET request for a collection, you will only get the n first results.

You can go to the next page using:

```
boolean hasNext = cytomine.nextPage(collection)
```

Full example (in Groovy):

```
cytomine.setMax(8);
projects = cytomine.getProjects()
boolean stop = false
while(!stop) {
    println "This page has projects " + projects.list.collect{it.id}.join(",")
    stop = !cytomine.nextPage(projects)
}
```

## Cytomine.java

The Cytomine class is the main class of the Cytomine java client.

For most of the ressource API, it provides CRUD methods like addUser, deleteAnnotation, getProjects,....

There are 5 private methods:

```
private <T extends Model> T fetchModel(T model)
private <T extends Collection> T fetchCollection(T collection)
private <T extends Model> T saveModel(T model)
private <T extends Model> T updateModel(T model)
private void deleteModel(Model model)
```

All of them take a Model (Annotation, User,...) as parameters and perform the corresponding HTTP action. These method use the toURL() from the previous section.

We can implement in Cytomine class all the method available in the client.
For the User part:

```
...
public User addUser (String username, String password, String firstname, String
lastname) throws CytomineException;
public User updateUser (String username, String password, String firstname, String
lastname) throws CytomineException;
public User deleteUser(int idUser) throws CytomineException;
public Project getUser (int id) throws CytomineException;
public UserCollection getUsers() throws CytomineException;
public UserCollection getUsersByProject(int idProject) throws CytomineException;
...
```

In case of failure, all these method throws a CytomineException. This class provides an HTTP code (401, 403, 404, …) and a message.

The addUser method builds a User and calls the saveModel method. The saveModel uses the toURL() method from the model and perform a POST action with the JSON from the user model.

```
public User addUser (String username, String password, String firstname, String
lastname) throws CytomineException {
    User user = new User()
    user.set("username",username)
    user.set("password",password)
    user.set("firstname",firstname)
    user.set("lastname",lastname)
    return saveModel(user) // saveModel = POST on /api/user.json
}
```

The updateUser and deleteUser use respectively updateModel and deleteModel. For these method an id is set so the URL is /api/user/$id.json.

If you want to retrieve a domain (list) you can use fetchModel (fetchCollection).

```
public Project getUser(int id) throws CytomineException{
    User user = new User()
    user.set("id",id)
    return fetchModel(user)
}

public UserCollection getUsers() throws CytomineException{
    UserCollection users = new UserCollection()
    return fetchCollection(users)
}

public UserCollection getUsersByProject(int idProject) throws CytomineException{
    UserCollection users = new UserCollection()
    users.addFilter("project",idProject);
    return fetchCollection(users)
}
```

## How to modify the Java-client

Requirements:

- Maven 2+
- JDK 1.7+

If you want to build the jar (Jar are available under target/ directory):

```
mvn package
```

Clean:

```
mvn clean
```

## Authors/License

ROLLUS Loïc
HOYOUX Renaud
STEVENS Benjamin

See LICENSE file.