May 14, 2015

INDIGO ELN COMPOUND REGISTRATION AND SEARCH SERVICE INSTALLATION GUIDE AND API REFERENCE

VERSION 1.2

GLOBAL HEADQUARTERS

41 University Drive, Suite 202
Newtown, PA 18940 USA

EU HEADQUARTERS

Corvin Offices I. Futó Street 47-53
Budapest, H-1082, Hungary

CIS HEADQUARTERS

9th Radialnaya Street, Building 2
Moscow, 115404, Russia

CONTENTS

1. INTRODUCTION 1
2. REQUIRED SOFTWARE 1
2.1. Database Server 1
2.2. Application Server 1
2.3. Client Side 1
3. REQUIRED HARDWARE 1
3.1. Database Server 1
3.2. Application Server 2
4. INSTALLATION OF SERVICE 2
4.1. Prerequisites 2
4.2. Installation 2
4.2.1. Installing Package 2
5. SERVICE OVERVIEW 3
5.1. Database Schema 3
5.2. Client Side 3
5.3. Registration Service 3
5.3.1. Usage of Token Hash 4
5.3.2. CompoundInfo Class 4
5.4. Search Service 4
5.4.1. Search Methods 5
5.4.2. Get Methods 5
5.4.3. FullCompoundInfo Class 5
6. UNINSTALLATION OF SERVICE 6
6.1. Removing Database Schema 6

1. INTRODUCTION

Compound Registration and Search service (CRS) registers compounds and batches by saving them to the Database and assigning a unique ID to them, searches compounds and batches by structure or ID or other parameters.

2. REQUIRED SOFTWARE

2.1. DATABASE SERVER

The following software is required for the database server:

- Oracle Database 10+
- Oracle Database Express Edition 10+
- PostgreSQL 9.2
- GGA Software Bingo 1.7+

You can read about Bingo cartridge installation here.

2.2. APPLICATION SERVER

The following software is required for the application server:

- Java SE 1.6+
- Apache Tomcat 7+
- Apache Maven 3.0+ (for build)

2.3. CLIENT SIDE

The following software is required on the client side:

- Java SE 1.6.

3. REQUIRED HARDWARE

3.1. DATABASE SERVER

Component	Supported configuration
Processor	Intel IA-32, Intel EM64T (Xeon), or AMD64 (Opteron) 1 GHz or higher Tip: A faster processor delivers faster performance.
RAM	1 GB or higher Tip: More RAM provides faster performance.
Hard disk space	10 GB
Network	Ethernet

3.2. APPLICATION SERVER

Component	Supported configuration
Processor	Intel IA-32, Intel EM64T (Xeon), or AMD64 (Opteron) 1 GHz or higher Tip: A faster processor delivers faster performance.
RAM	1 GB or higher Tip: More RAM provides faster performance.
Hard disk space	1 GB
Network	Ethernet

4. INSTALLATION OF SERVICE

4.1. PREREQUISITES

- Ensure that Oracle (PostgreSQL) and Tomcat servers are up and you have Write access to the Tomcat's webapps folder.
- Check the presence of mvn and javac in the system PATH.
- Check the presense of sqlplus (for Oracle) or psql (for PostgreSQL) in the system PATH.

4.2. INSTALLATION

4.2.1. Installing Package

To install the CRS package:

1. Unzip the archive to any location. The subsequent instructions presume paths relative to the folder of the extracted archive.
2. Set the properties of the package. They are located in the **src/main/resources/crs.properties** file. You need to set the following values:

Option key	Description of the option	Example of the value
DATABASE_DRIVER_CLASS	Driver class name for database connection	org.postgresql.Driver
DATABASE_URL	Full database JDBC url	jdbc:postgresql://localhost:5432/postgres
DATABASE_USERNAME	Database user	crs
DATABASE_PASSWORD	Password of the database user	crscrscrscrs

3. Install the database schema. You need to execute the following command in the **database** folder (change the database url and admin username to those used in your database configuration):

For Oracle:

```
$ sqlplus sys/admin@//localhost:1521/xe as sysdba @oracle/schema_install.sql
```

For PostgreSQL:

```
$ psql -h localhost -p 5432 -d postgres -U postgres -f
postgresql/schema_install.sql
```

4. Build the package:

```
$ mvn clean package
```

5. Deploy the package to the application server.
6. Add users to the database. You need to do it directly from the database console (using the sqlplus utility or SQLDeveloper application):

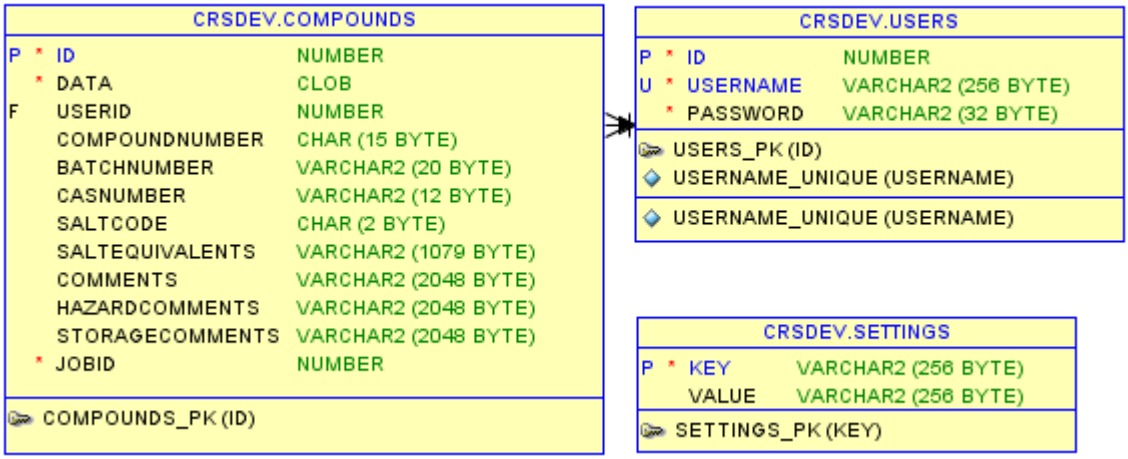
```
> insert into users(username, password) values ('test', 'test');
```

The password is automatically hashed. You can add as many users as you need.

5. SERVICE OVERVIEW

5.1. DATABASE SCHEMA

The following picture shows the database schema:



5.2. CLIENT SIDE

After building CRS, the crs-client.jar is generated in the **target** folder. It contains the CrsClient class that allows invoking services. The CRS Client depends on **spring-web:3.x** and **commons-httpclient:3.x**.

5.3. REGISTRATION SERVICE

The Registration service has the following web methods:

Method	Description
`String getTokenHash(String username, String password)`	Authorizes the user and returns the token hash to use in the next call of the method.
`long submitForRegistration(String tokenHash, CompoundInfo compoundInfo)`	Submits a compound for registration. Returns the ID of a submitted job that could be useful for checking the registration status.

Method	Description
`long submitListForRegistration(String tokenHash, List<CompoundInfo> compoundInfoList)`	Submits the list of compounds for the registration. Returns the ID of a submitted job that could be useful for checking the registration status.
`CompoundRegistrationStatus checkRegistrationStatus(String tokenHash, long jobId)`	Checks the registration status of a selected job.
`boolean isUnique(String tokenHash, String compound)`	Checks the presence of a compound in the database. The compound may be in SMILES, CML, or Molfile formats.

5.3.1. Usage of Token Hash

You need to provide Token Hash to most methods. But before that, you need to get the token hash for the user and password combination by calling the method getTokenHash(). There can be only one valid token hash for each user, and getTokenHash() removes old tokens for users when the server calls getTokenHash() for this user. If an old token hash is used, the checkRegistrationStatus() method returns WRONG_TOKEN_DURING_REGISTRATION or WRONG_TOKEN_DURING_CHECK.

5.3.2. CompoundInfo Class

The CompoundInfo class stores information about a compound; its instances are passed to the registration service methods. The CompoundInfo class has the following fields:

Data	Compound structure
batchNumber	NBK batch number, unique for each batch, consists of 'notebook number-experiment number-batch number'
casNumber	CAS number
saltCode	Code of the salt that is part of the compound
saltEquivalents	Number of the salt equivalents
Comments	Comments
hazardComments	Description of the hazard handling conditions
storageComments	Description of the hazard conditions
stereoIsomer	Description of the stereochemistry of the compound

All fields have getter and setter methods.

5.4. SEARCH SERVICE

The full name of the search service is BingoSearchService.

The Search service uses the benefits of the EPAM Bingo cartridge for searching the compounds. Bingo search procedures have optional parameters that can tune the search. You can find detailed information about Bingo procedures and options here.

5.4.1. Search Methods

Each method returns a list of IDs of the found compounds. You can get the compound information of the found molecules by the getCompoundById method.

Method	Description
`List<Integer> searchExact(String compound, String options)`	Exact search for a compound in the database with selected query options.
`List<Integer> searchSub(String compound, String options)`	Substructure search for a compound in the database with selected query options.
`List<Integer> searchSmarts(String compound)`	SMARTS search in the database with selected query options.
`List<Integer> searchSim(String compound, String metric, Double bottomValue, Double topValue)`	Similarity search in the database with selected query options. Similarity metric can be “Tanimoto”, “Tversky”, or “Euclid-sub” with the values between 0 and 1.

5.4.2. Get Methods

The Get methods can return `List<String>` for a single compound or `List<String[]>` for several compounds. These strings come in the following order: data, `compoundNumber`, `batchNumber`, `casNumber`, `saltCode`, `saltEquivalents`, `comments`, `hazardComments`, and `storageComments`.

Method	Description
`List<String> getCompoundById(int id)`	Get compound data by a selected ID in the string list format.
`List<String> getCompoundByNumber(String compoundNumber)`	Get compound data by a compound number.
`List<String> getCompoundByCasNumber(String casNumber)`	Get compound data by a CAS number.
`List<String[]> getCompoundByJobId(String jobId)`	Get the list of compound data by a Job ID.
`List<String> getCompoundByBatchNumber(String batchNumber)`	Get compound data by a batch number.

5.4.3. FullCompoundInfo Class

The FullCompoundInfo class is used to store information about registered compounds. It has all fields of the CompoundInfo class, and the following two fields:

Field	Description
compoundNumber	Compound ID, a number assigned to a compound after registration. Each compound has its own number. All batches of this compound have the same number, created from the conversational batch number.

Field	Description
conversationalBatchNumber	A number assigned to a compound after registration. Each compound has its own number. All batches of this compound have the same number and reference to the batch quantity. Salt code is reflected in the conversational batch number. The format is xxxxxxxx-yy-zzz, where xxxxxxxx is the compound ID (compound number), yy is the salt code (00-parent structure), and zzz is the batch quantity.

These fields have both getter and setter methods.

6. UNINSTALLATION OF SERVICE

6.1. REMOVING DATABASE SCHEMA

If you need to remove all information that was written to the database by the services, you can run the following command in the **database** folder (change the database url and admin username for you database configuration):

For Oracle:

```
$ sqlplus sys/admin@//localhost:1521/xe as sysdba
@oracle/schema_uninstall.sql
```

For PostgreSQL:

```
$ psql -h localhost -p 5432 -d postgres -U postgres -f
postgresql/schema_uninstall.sql
```

Be careful! This command removes all data from the database, including all uploaded compounds.