Celerio

Reference Documentation

Jaxio

Celerio: Reference Documentation

Jaxio

Version 3.0.64

Abstract

Celerio Version 3.0.64 Reference Documentation

Legal notice

Table of Contents

List of Tables

List of Examples

Preface

There are so many Java technologies around! It is moving so fast! Which ones to choose? And then, what are the associated best practices? You do not always have the time to answer these questions.

Celerio's mission is to answers these questions through code generation.

Celerio's benefits include applications development productivity boost and maintenance costs reduction.

Celerio is a code generator developed by Jaxio . Its usage requires a commercial license.

Chapter 1. Introduction

Celerio

Celerio generates the foundations of a Java Web application built on standard technologies. Celerio may be used for different purposes: application migration, new project development, technical training, etc. Celerio is distributed as a Maven 2 plugin which greatly eases its integration in your project development.

Celerio itself is written in Java. It executes generation templates that produce some source code: Java files, XML files, etc. The generation templates are written in Velocity . Jaxio provides ready-to use templates so you can focus on real business needs but you may also creates your own generation templates.

Celerio uses an entity-relationship model as a starting point. This model can be produced by any tool as long as the tool respects Celerio's expected format. In practice Jaxio provides a tool that extracts an existing database schema to produce the initial model. Of course this entity-relationship model can be refined by configuration.

Celerio is composed of three main components: a database schema extractor tool, the core Celerio Engine and some template packs.

Database Schema Extractor tool

This tool reverses an existing database schema, using the JDBC Meta Data API. It connects to your database, and extracts the various meta-information such as tables' name, columns' name, constraints, relations, comments, etc. The extraction result is written in an XML file. This tool runs in a read only mode, it does not read your real data or modify your database.

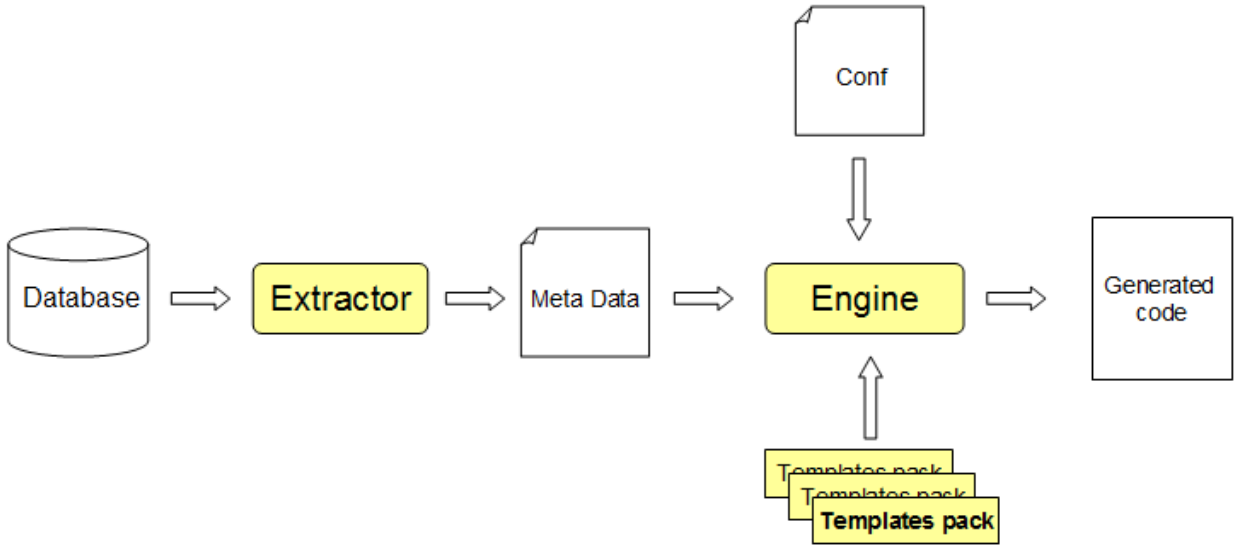
Celerio Engine

Celerio Engine loads the raw entity-relationship model extracted from your database and a configuration file which instructs Celerio how to refine this model. Indeed, the extracted model does not carry all the meta information, it is up to you to enrich it. For example, through configuration, you may add inheritance to your model. From this enriched model, Celerio constructs a graph into memory and let the generation templates access to it when it executes them.

Template packs

The generation templates that Celerio executes are grouped into jar files called templates packs. Celerio loads all the template packs found on the classpath, so make sure you add them as jar dependencies when executing Celerio. By default Celerio executes all the templates found in a pack, but this can be configured to filter out certain templates. The templates packs are regularly updated to take into account the

latest industry standards and practices. The current template packs generate some source code that uses the following frameworks or standards: SpringFramework, JPA, Hibernate, Bean Validator, Spring Web Flow, RichFaces, etc.



How Celerio works

The generated source code follows industry standards and best practices. It is organized into a project structure following the Maven conventions.

Celerio also uses some conventions. If you follow them, you can reduce the amount of configuration.

The Database Schema Extractor and Celerio Engine are distributed as Maven 2 plugins, which simplifies their installation and make them compatible with IDE that already provide Maven 2 integration. Eclipse, Netbeans and IntelliJ have outstanding Maven support.

Celerio allows you to replace, skip, modify or extend the generated source code allowing you to use Celerio in an iterative development.

Last but not least, Celerio is not required at runtime. There is no lock-in.

When to use Celerio?

Celerio targets applications that use a relational database and whose database schema is close to the domain layer. From there Celerio may be used for different purposes.

Application Rewriting

Deciding to rewrite an application in Java is a tough decision. There are often many risks. However, when your end users ask for patches and features that you cannot deliver in time because of some some

old code that nobody understand anymore, you do not have many choices. But it is too much work, you would need a second team or more time, but there is no budget for that. Celerio helps your existing team to make the first move toward your application rewriting and support your team during the application development cycle.

Rapid prototyping

Writing a prototype for free is a common practice in the industry. You have to work fast, for free, and deliver top notch quality to get the deal. Celerio increases drastically your chances to succeed, it gives you the possibility to prototype extremely fast with no compromise on quality and architecture.

New application

Writing an application from scratch is also possible with Celerio. Instead of designing your domain layer in Java, you directly design you database schema.

Spread your best practice and architecture

New templates packs can be easily developed to take into account your in-house libraries and company best practices. Once these packs are ready, they can be deployed as any other jar in your private Maven repository.

Training

Finally, the code generated by Celerio may be used as training material. The code is readable, well organized, unit tested and documented. Developers should learn from it.

What to expect from Celerio?

You can expect a 30% gain on your application development.

However it is important to understand that Celerio is not here to replace your developer's brain. So do not count on Celerio to do all the smart job. Instead Celerio is here to let your developers focus on the real value of your final application. Celerio automates the tedious and repetitive task of writing the backbone of your Java project.

Not smart, but not dumb either, Celerio goes way beyond a simple copy paste. It covers subject such as persistence, inheritance, 2d level cache, security, dependency injection, distribution, localization, unit tests, etc.

Chapter 2. Installation

JDK 6

Celerio and the generated code require JDK 6 to compile and run. *Download and install the latest JDK from http://java.sun.com*

Once installed, make sure you can run `javac` from the command line. Open a command line and run:

javac –version

You should get a result similar to this:

```
javac 1.6.0_18
```

Maven 2

Celerio and the generated code require Maven 2.

Download and install Maven 2 from http://maven.apache.org

Once installed, make sure you can run Maven 2 from the command line. Open a command line and run:

mvn –v

You should get a result similar to this:

```
Maven version: 2.0.9
Java version: 1.6.0_18
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
```

Source control system

Celerio leverages source control (SVN and CVS) usage to provide features allowing the user to take control over certain generated file.

Before generating such file, Celerio always check if a file of the same name exists on disk and is present under source control.

Celerio assumes that files under source control should not be overwritten arbitrarily.

To really take advantage of Celerio's collision features, you must use a source-control system for the project you intent to generate with Celerio.

Celerio

Celerio is distributed as a Maven 2 plugin. Celerio plugin is hosted on Jaxio Maven 2 repository. You can either directly uses it if your organization allows access to Jaxio's repository or deploy it on your organization Maven's repository. Please contact Jaxio's for further details.

Chapter 3. Start a new project

Bootstrap the skeleton

To create a new project skeleton, invoke the `bootstrap` goal of the `maven-bootstrap-plugin` from a command line as follow:

mvn com.jaxio.celerio:maven-bootstrap-plugin:3.0.64:bootstrap

When prompted, enter your application name and the desired root package and select the desired type of application you want to generate (a SpringMVC project, a JSF/WebFlow project etc...).

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [com.jaxio.celerio:maven-local-bootstrap-plugin:3.0.42-SNAPSHOT:bootstrap] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [local-bootstrap:bootstrap {execution: default-cli}]
[INFO]
[INFO]        _______         __                 _
[INFO]       .' ___  |       [  |               (_)
[INFO]      / .'   \_| .---.  | | .---.  _ .--.  __   .--.
[INFO]      | |       / /__\\ | |/ /__\\[ `/'`\][  |/ .'`\ \
[INFO]      \ `.___.'\| \__., | || \__., | |     | || \__. |
[INFO]       `._____.' '.__.'[____]'.__.'[____]   [____]'.__.'    v3.0.42-SNAPSHOT
[INFO]
[INFO]        (c) 2005-2011 Jaxio, http://www.jaxio.com
[INFO]        Please report bugs/ideas/feedbacks to support@jaxio.com
[INFO]

Enter your application name: [bootstrap]
myapp

Enter the Java root package of your application: [com.jaxio.demo.myapp]

Enter the type of application you want to generate:
 1) [work in progress] JSF2, JPA2, with Spring WebFlow 2.2.1 and PrimeFaces 2.2.RC2
 2) [New] Backend only project with JPA 2 and Spring 3
 3) Application Java/JEE/Spring3/SpringMVC
 4) Ext-GWT
```

As soon as you select the type of application, it creates a new folder named after your application name and copies a minimal set of files required to run Celerio. It also output on the console the command you should run.

```
[INFO] ============================================
[INFO]  PLEASE EXECUTE THE FOLLOWING COMMANDS:
```

```
[INFO]
[INFO]   cd myapp
[INFO]   mvn -Ph2,db,metadata,gen jetty:run
[INFO]
[INFO]  Then open the page http://localhost:8080/
[INFO]
[INFO] ========================================
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9 seconds
[INFO] Finished at: Thu Jan 27 18:09:23 CET 2011
[INFO] Final Memory: 16M/81M
[INFO] ------------------------------------------------------------------------
```

Skeleton files

In general, the minimal files include at least:

- `pom.xml` : A complete Maven2 POM file.
- `src/main/sql/h2/01-create.sql` : an SQL script to create, init and drop a local H2 database. This is an example database that Celerio is going to reverse.
- `src/main/config/maven-celerio-plugin/maven-celerio-plugin.xml` : A Celerio's configuration file.

Reverse, generate and run

Convention over configuration

Maven 2 by convention expects certain files to be present under certain location. For example all resource files are expected to be under src/main/resources. Celerio follows these conventions to generate the various source code and configuration files. As an immediate benefit, you can rely entirely on Maven 2 to build the project generated by Celerio.

You are now ready to

- create the database out of the sample SQL script provided,
- reverse this database schema and generate the corresponding source code and configuration files,
- build the generated project,

- and run the resulting web application!

To do so, from your project folder execute the Maven command printed on the console, for example:

C:\myapp>mvn –Ph2,db,metadata,gen jetty:run

> ### Note
>
> Please note that this command may take a while to complete the first time as Maven needs to download all the necessary plugins and dependencies to build and run the project.

The `-P` option stands for 'profile'. In the above command, we activate the 'db' profile to create a new database, the 'metadata' profile to reverse it and the 'gen' profile to generate the corresponding source code. All these profiles are declared in the main `pom.xml` file. The `jetty:run` starts an embedded web server.

Now you can try out your new web application. Open http://localhost:8080/ in your browser. Can you log in as admin/admin? If yes, congratulations!, all went well...

Unit Test

Celerio also generates some unit tests. To make sure they all pass, simply run:

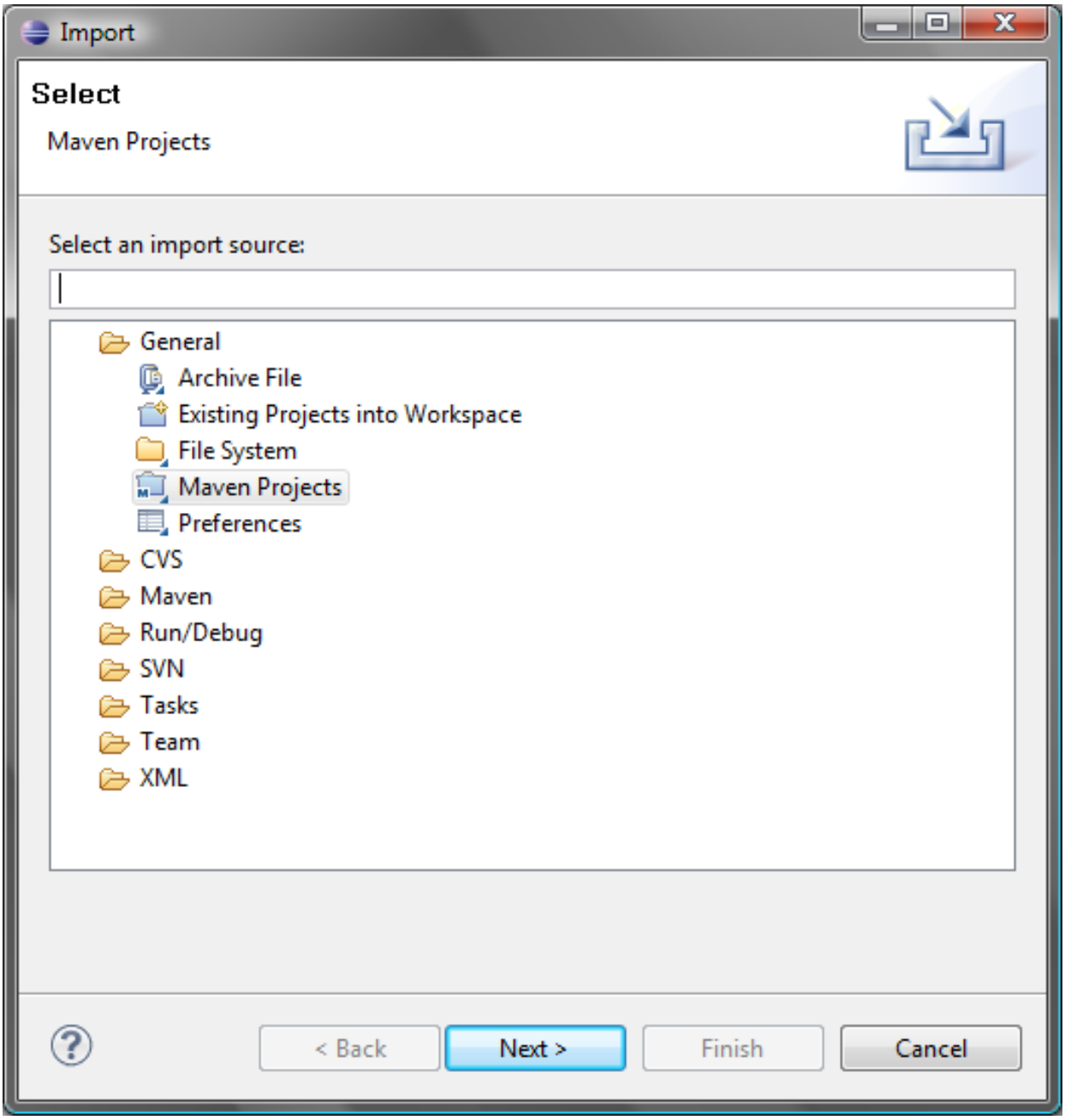
C:\myapp>mvn test

As you can notice here, we have not set any profile since there is no need to re-extract the database meta data or re-generate the source code.

Import in Eclipse

> ### Note
>
> We assume here that you have already configured Eclipse's Maven plugin. If not please refer to http://m2eclipse.sonatype.org/.

You can now import your first generated project into Eclipse. Make sure to import it as an *existing Maven project.*



Eclipse Maven 2 - Import as Maven project

Important

Once imported please add manually to your build path the following Java folders

- `src/main/generated-java`
- `src/test/generated-java`

To do so, from Eclipse, simply right-click on each folder and select the menu 'Build Path' ->

'Use as source folder'.

Run Celerio from Eclipse

Since Celerio is a Maven plugin, you can invoke it as any other Maven plugin from Eclipse. Right-click on your project and select the 'Run As' -> 'Run Configurations...' menu. From there you can set the Maven command to invoke and save this setting in a `.launch` file at the root of your project. This 'Run configuration' will appear in your top level menu.

bootstrap:bootstrap

Full name

`com.jaxio.celerio:maven-bootstrap-plugin:3.0.42-SNAPSHOT:bootstrap`

Description

This plugin creates a default project folder layout following Maven conventions. A Maven `pom.xml` is generated as well as default files to help you start from scratch a project. These files represent the minimum required files to produce a project using Celerio.

Dependency

- Since version: `3.0.0`.
- Binds by default to the lifecycle phase: `initialize`.

Attributes

- `defaultBootstrapPackName` : The bootstrap pack to use by default if none is specify or if running in non-interactive mode.
- `interactive` : Will the bootstrap ask interactively for appName and rootPackage?

Table 3.1. `bootstrap:bootstrap` plugin attributes list

Name	Type	Expression	Default
`defaultBoot-`	`String`	`${maven-bootstra`	`pack-`

Name	Type	Expression	Default
strapPackName		p-plu-gin.defaultBoots trapPackName}	jsf-bootstrap
interactive	String	${maven-bootstra p-plu-gin.interactive}	true

Chapter 4. Database Schema extraction

Principle

The database schema extraction is a pretty basic step performed by the `maven-db-metadata-plugin`, a Maven 2 plugin provided by Jaxio.

This plugin connects to your database, extracts all the valuable meta data information, and dumps them into an XML file. That's all.

> **Note**
>
> The metadata file produced by the plugin is not meant to be edited manually. If you need to change some settings, you should either change your database schema or use Celerio's configuration file.

While pretty basic the extraction step may take several minutes on large database schema (e.g. more than 200 tables). *To prevent useless extraction* , the simplest approach is to declare a Maven profile dedicated to database extraction and activate it only when needed. As an example you can use the `pom.xml` that Celerio `bootstrap` goal produces.

dbmetadata:extract-metadata

Full name

`com.jaxio.celerio:maven-dbmetadata-plugin:3.0.42-SNAPSHOT:extract-metadata`

Description

This plugin connects to a relational database using JDBC and reverses the database schema meta data. The reverse engineering consists in serializing the information returned by the JDBC driver into an XML file (see the documentation of the java.sql.DatabaseMetaData for more information). The `metadata.xml` file produced by this plugin is used by Celerio's generate goal. Please refer to the the section called “celerio:generate” .

Dependency

- Since version: `3.0.0`.

- Binds by default to the lifecycle phase: `generate-sources`.

Attributes

- `jdbcCatalog` : Specify the JDBC catalog.
- `jdbcDriver` : Specify the JDBC driver class. Example: `org.postgresql.Driver`
- `jdbcOracleRetrieveRemarks` : Should the Oracle remarks be retrieved ? Please note that this will impact the speed of the reverse engineering of your database.
- `jdbcOracleRetrieveSynonyms` : Should the synonyms be retrieved ?
- `jdbcPassword` : Specify the JDBC password.
- `jdbcSchema` : Specify the JDBC schema.
- `jdbcUrl` : Specify the JDBC url to connect to your database. Make sure that you connect with enough privileges to access the meta data information. Example: `jdbc:h2:~/.h2/sampledatabase`
- `jdbcUser` : Specify the JDBC user, this user needs to have the privilege to access the database metadata.
- `skip` : Should the database meta data extraction be skipped ? This is a common pattern in Maven, where you can skip plugins using profiles to fully adapt your build.
- `targetFilename` : The fully qualified name of the XML file created by this plugin.

Table 4.1. `dbmetadata:extract-metadata` plugin attributes list

Name	Type	Expression	Default
`jdbcCatalog`	`String`	`${jdbc.catalog}`	
`jdbcDriver`	`String`	`${jdbc.driver}`	
`jdbcOracleRetrieveRemarks`	`boolean`	`${jdbc.oracleRetrieveRemarks}`	`false`
`jdbcOracleRetrieveSynonyms`	`boolean`	`${jdbc.oracleRetrieveSynonyms}`	`true`
`jdbcPassword`	`String`	`${jdbc.password}`	
`jdbcSchema`	`String`	`${jdbc.schema}`	
`jdbcUrl`	`String`	`${jdbc.url}`	

Name	Type	Expression	Default
`jdbcUser`	`String`	`${jdbc.user}`	
`skip`	`boolean`	`${maven-celerio-plugin.skip}`	`false`
`targetFilename`	`String`	`${maven-metadata-plugin.targetFilename}`	`${basedir}/src/main/config/maven-celerio-plugin/metadata.xml`

Examples

With explicit configuration

To use `dbmetadata` plugin in your Maven build, add the plugin definition with the JDBC information to access the database from which you want to extract the meta data.

```
<plugin>
  <groupId>com.jaxio.celerio</groupId>
  <artifactId>maven-dbmetadata-plugin</artifactId>
  <version>3.0.11</version>
  <executions>
    <execution>
      <id>Extract the database schema.</id>                    ❶
      <goals>
        <goal>extract-metadata</goal>
      </goals>
    </execution>
    <configuration>                                            ❷
      <jdbcUrl>jdbc:h2:~/.h2/sampledatabase</jdbcUrl>
      <jdbcUser>myuser</jdbcUser>
      <jdbcPassword>myuser</jdbcPassword>
    </configuration>
  </executions>
  <dependencies>
    <dependency>                                               ❸
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>1.2.125</version>
    </dependency>
  </dependencies>
</plugin>
```

❶ The `id` element is mostly used for documentation as it is displayed during the build process.
❷ Specification of the configuration by stating explicitly all the parameters

❸ Please note that you need to specify the dependency for your JDBC driver to the plugin.

Important

The `jdbcUser` must have enough privileges to access the database meta-information. Otherwise Celerio cannot reverse the database and generate the code.

With maven properties

The metadata plugin can get its data from the maven properties.

This allows you to keep the plugin configuration short and reuse the `jdbc.user` property elsewhere, for example when working with the `sql-maven-plugin` to recreate your database schema.

Important

Using properties is really convenient so you can switch configuration using maven profiles

You can refer to the list of configuration elements. .

Celerio can get its jdbc information from your current properties

```
<properties>                                    ❶
  <jdbc.driver>org.h2.Driver</jdbc.driver>
  <jdbc.url>jdbc:h2:~/.h2/sampledatabase</jdbc.url>
  <jdbc.user>myuser</jdbc.user>
  <jdbc.password>myuser</jdbc.password>
</properties>
<build>
  <plugins>
    <plugin>
      <groupId>com.jaxio.celerio</groupId>
      <artifactId>maven-dbmetadata-plugin</artifactId>
      <version>3.0.0</version>
      <executions>
        <execution>
          <id>Extract the database schema.</id>
          <goals>
            <goal>extract-metadata</goal>
          </goals>
        </execution>
      </executions>                             ❷
      <dependencies>
        <dependency>
          <groupId>com.h2database</groupId>
          <artifactId>h2</artifactId>
          <version>1.2.125</version>
        </dependency>
      </dependencies>
    </plugin>
```

```
  </plugins>
</build>
```

❶ Specify maven properties, they will be picked-up by the metadata plugin.
❷ No need of configuration, as all the required properties are available as maven properties.

This configuration has the same effect as the previous example.

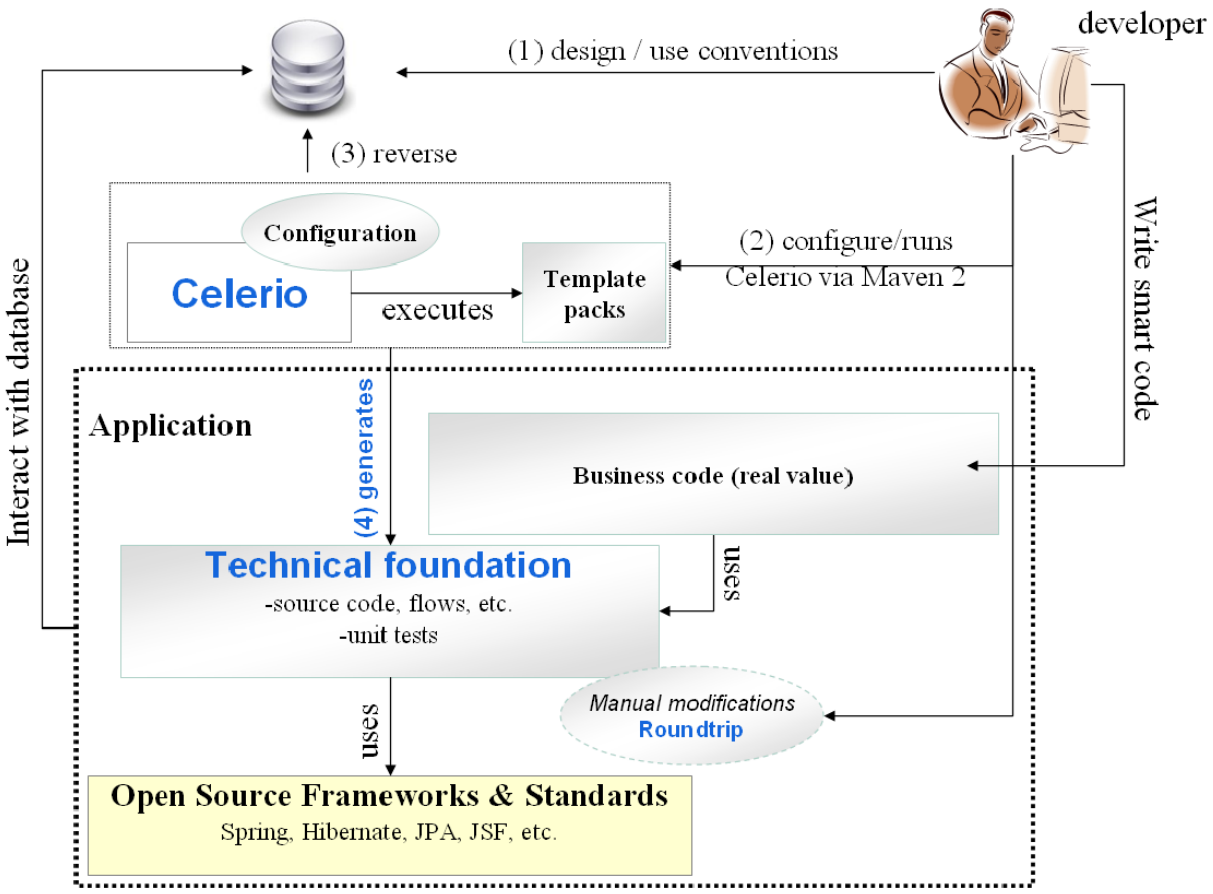
Chapter 5. Code generation

Principle

The code generation step is performed by the `generate` goal of the `maven-celerio-plugin` provided by Jaxio.

The code generation step is driven by several factors:

- *The database meta data:* It serves as a main entry model. The various column's constraints such as `NOT NULL` , the foreign keys, etc. are exploited by Celerio in the code generation process.
- *Celerio conventions:* Please refer to Chapter 7, *Conventions and integration*.
- *Celerio configuration:* Please refer to Chapter 9, *Configuration*.
- *Celerio template packs:* Please refer to ???.
- *Generated code modifications:* Please refer to Chapter 6, *Code modification, regeneration and collisions*.



Source code generation is driven by several factors

As for database schema extraction, the code generation step does not need to be performed at each build. *To prevent useless generation* , the simplest approach is to declare a Maven profile dedicated to code generation. As an example you can use the `pom.xml` that the Celerio `bootstrap` goal produces.

celerio:generate

Full name

`com.jaxio.celerio:maven-celerio-plugin:3.0.42-SNAPSHOT:generate`

Description

The core Celerio Engine is invoked by this plugin. This plugin can either connect directly to a database and extract the metadata information or use the metadata.xml file produced by the maven-db-metadata-plugin:extract-metadata goal. Please refer to the section called “dbmetadata:extract-metadata”.

Dependency

- Since version: `1.0.0`.
- Binds by default to the lifecycle phase: `generate-sources`.

Attributes

- `baseDir` : The current folder
- `jdbcCatalog` : Specify the JDBC catalog.
- `jdbcDriver` : Specify the JDBC driver. Example: `org.postgresql.Driver`
- `jdbcOracleRetrieveRemarks` : Should the Oracle remarks be retrieved ? Please note that this will impact the speed of the reverse engineering of your database.
- `jdbcOracleRetrieveSynonyms` : Should the synonyms be retrieved ?
- `jdbcPassword` : Specify the JDBC password.
- `jdbcSchema` : Specify the JDBC schema.
- `jdbcUrl` : Specify the JDBC url. Example: `jdbc:h2:~/.h2/sampledatabase`
- `jdbcUser` : Specify the JDBC user, this user needs to have the privilege to access the database metadata.
- `outputDirectory` : The output folder.
- `project` : Maven project, this is by default the current maven project.
- `skip` : Should the source code generation be skipped ? This is a common pattern in Maven, where you can skip plugins using profiles to fully adapt your build.
- `xmlConfiguration` : The relative path to the Maven Celerio configuration file. The default value is `src/main/config/maven-celerio-plugin/maven-celerio-plugin.xml`
- `xmlMetadata` : The relative path to the metadata.xml file produced by the maven-db-metadata-plugin:extract-metadata goal. If this file exists it will be used, otherwise Celerio will access the database directly. The main purpose of this file is to speed-up the generation process, as for large database schema the reverse engineering takes time. An other very important benefit of this feature is to store the file in your source control, thus having a reproducible build.
- `xmlTemplatePacksOverride` : The relative path to a Maven Celerio configuration file dedicated to override the template packs definition present in the main xml configuration file. This configuration file is useful when working on multi-modules project. Indeed you can set for each module exactly the template packs that should be used. Keep in mind that only the template packs definition will

be extracted from this file. The default value is `src/main/config/maven-celerio-plugin/celerio-template-packs.xml`

Table 5.1. `celerio:generate` plugin attributes list

Name	Type	Expression	Default
baseDir	String	${basedir}	
jdbcCatalog	String	${jdbc.catalog}	
jdbcDriver	String	${jdbc.driver}	
jdbcOracleRetrieveRemarks	boolean	${jdbc.oracleRetrieveRemarks}	false
jdbcOracleRetrieveSynonyms	boolean	${jdbc.oracleRetrieveSynonyms}	true
jdbcPassword	String	${jdbc.password}	
jdbcSchema	String	${jdbc.schema}	
jdbcUrl	String	${jdbc.url}	
jdbcUser	String	${jdbc.user}	
outputDirectory	String	${maven-celerio-plugin.directory}	${basedir}
project	MavenProject	${project}	
skip	boolean	${maven-celerio-plugin.skip}	false
xmlConfiguration	String	${maven-celerio-plugin.configuration}	${basedir}/src/main/config/maven-celerio-plugin/maven-celerio-plugin.xml
xmlMetadata	String	${maven-celerio-plugin.xml.metadata}	${basedir}/src/main/config/maven-celerio-plugin/metadata.xml
xmlTemplatePack-	String	${maven-celerio-	${basedir}/src/m

Name	Type	Expression	Default
sOverride		plu- gin.packs.config uration}	ain/con- fig/ maven-cel- erio-plu- gin/cel- erio-tem- plate-packs.xml

generate goal examples

With explicit configuration

To use metadata plugin in your maven build, add the plugin definition with the jdbc information to access the database from which you want to extract the model in xml.

```
<plugin>
  <groupId>com.jaxio.celerio</groupId>
  <artifactId>maven-celerio-plugin</artifactId>
  <version>3.0.0</version>
  <executions>
    <execution>
      <id>Generates files using the extracted database schema.</id> ❶
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
    <configuration>                                    ❷
      <jdbcUrl>jdbc:h2:~/.h2/sampledatabase</jdbcUrl>
      <jdbcUser>myuser</jdbcUser>
      <jdbcPassword>myuser</jdbcPassword>
    </configuration>
  </executions>
  <dependencies>
    <dependency>                                       ❸
      <groupId>com.jaxio.celerio.packs</groupId>
      <artifactId>backend</artifactId>
      <version>3.0.0</version>
    </dependency>
    <dependency>                                       ❹
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>1.2.125</version>
    </dependency>
  </dependencies>
</plugin>
```

❶ The `id` element is mostly used for documentation as it is displayed during the build process.

❷ Specification of the configuration by stating explicitly all the parameters
❸ Specify the dependency on the packs to be used when producing the files.
❹ Please note that you need to specify the dependency for your jdbc driver to the plugin.

With maven properties

```
<properties>                                    ❶
  <jdbc.driver>org.h2.Driver</jdbc.driver>
  <jdbc.url>jdbc:h2:~/.h2/sampledatabase</jdbc.url>
  <jdbc.user>myuser</jdbc.user>
  <jdbc.password>myuser</jdbc.password>
</properties>
<build>
  <plugins>
    <plugin>
      <groupId>com.jaxio.celerio</groupId>
      <artifactId>maven-celerio-plugin</artifactId>
      <version>3.0.0</version>
      <executions>
        <execution>
          <id>Generates files using the extracted database schema.</id>
          <goals>
            <goal>generate</goal>
          </goals>
        </execution>
      </executions>                             ❷
      <dependencies>
        <dependency>
          <groupId>com.jaxio.celerio.packs</groupId>
          <artifactId>backend</artifactId>
          <version>3.0.0</version>
        </dependency>
        <dependency>
          <groupId>com.h2database</groupId>
          <artifactId>h2</artifactId>
          <version>1.2.125</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

❶ Specify maven properties, they will be picked-up by the metadata plugin.
❷ No need of configuration, as all the required properties are available as maven properties.

Generation logs

For each file generated, the Celerio `generate` goal writes a one-line log message giving status information. The status line has the following format: [<template pack>][<message>:<generated file path>]

The template pack is simply the template pack name.

The message indicates how the generation went; it could be of the form

- `GENERATE` : The target file does not exists yet, Celerio generates it normaly.
- `GENERATE AND REPLACE` : The target file is already present, it is not under user's control, and it is different from the one that Celerio wants to generate. Celerio replaces the file with its new version.
- `GENERATE IN COLLISIONS FODLER` : The target file is already present, it is under user's control, and it is different from the one that Celerio wants to generate. Celerio leaves the user's file where it is and generate its file's version in the collisions folder.
- `IDENTICAL EXISTS` : The file already exists and is indentical to the file that should have been generated. Celerio does not replace it to preserve the timestamp of the original file and for obvious performance reasons.
- `COPY` : The target file, a static resource such an image, does not exist yet. Celerio copy it normally.
- `COPY IN COLLISION FOLDER` : The target file is already present, it is under user control and it is different from the one that Celerio wants to copy. Celerio leaves the user's file where it is and copy its file's version in the collisions folder.

Chapter 6. Code modification, regeneration and collisions

Introduction

One of the most important feature of a code generator is its ability to preserve manual code modifications across re-generations. To solve this challenge, Celerio proposes various pragmatic solutions depending on the nature of the generated file.

Java files

By default, Celerio generates the main Java file under the directory `src/main/generated-java` .

In this section you will learn how to modify a generated Java file without loosing your modifications. The first approach consists in simply moving the file in a user reserved folder. The second approach involves inheritance of a base class. Both approaches have pros and cons.

> **Important**
>
> In any case, you should never modify directly the generated Java file as you will loose your changes as soon as you regenerate the code.

Code modifications through reserved folders

Before modifying the generated Java file, move it to the `src/main/java` folder. For example, if you want to edit the generated `BankAccount.java` file, move it from `src/main/generated-java/<package path>/BankAccount.java` to `src/main/java/<package path>/BankAccount.java`

> **Note**
>
> When you copy the file make sure you preserve the package directory structure.

Since all files under the `src/main/java` are considered as user reserved files, Celerio will not interfere with them. Next time you run Celerio, Celerio will detect the presence of BankAccount.java under `src/main/java/<package path>` , leave it untouched, and will not generate it here.

The main advantage of this approach is that, you now can do whatever you want with the file, in particular you can modify any method's body etc.

The main drawback may show up if your database schema changes. You may have to reflect those changes in the file that you now own as a regular file. Hopefully, to help you in this task, Celerio generates the file in a dedicated collision folder under the `target` folder. Please refer to the section called “Collisions and Merging” .

Code modifications through 'Base' inheritance

Celerio enables you to extend a generated Java class without modifying the generated Java code or the generated resources files.

Let’s take an example with the `Account` class which maps the `Account` table of your database schema.

Let's assume the `Account` source file is located at `src/main/generated-java/com/example/demo/domain/Account.java` It looks like this:

```
package com.example.demo.domain;
@Entity
@Table(name = "ACCOUNT")
public class Account implements Serializable {
   //...
   private String firstName;
   private String lastName;
   //... skipped for clarity
}
```

Let’s assume from a JSP you want to display, using a single method call, the full name of the user. One solution would be to add the following method to the Account class:

```
public String getFullName() {
      return getFirstName()+ " " + getLastName();
   }
```

Instead of adding this method in the generated file, where it would be mixed with generated code, Celerio supports a cleaner solution that enables you to extend the generated class of you choice and at the same time preserve the same class name!

This technique has two major benefits:

No need to modify the existing files (Java, xml, etc.) that reference the Java class that you want to extend.

The hand written code is isolated in a separated class, in a separated file, which greatly improves the clarity of your code.

Best way to understand how it works is to try it:

Create a new `Account.java` file and save it in the directory: `src/main/java/com/example/demo/domain`

Add this file under source control

Note

Note that the new path is almost the same as the old path. The only difference is that the file is now under `src/main/java` and not `generated-java` .

Open the newly created `Account.java` file and copy paste the following code:

```
package com.example.demo.domain;

@Entity
@Table(name = "ACCOUNT")
public class Account extends AccountBase implements Serializable {
   public String getFullName() {
      return getFirstName()+ " " + getLastName();
   }
}
```

This `Account` class now extends a class that does not exist yet!

Next time you run Celerio, it will detect your new `Account.java` file and will also detect that the `Account` class extends a class that has the same name plus the `Base` suffix.

This is enough for Celerio to assume that you want to impersonate the class, and that it should generate the code corresponding to Account table in a class called `AccountBase` instead of a class called `Account` .

The source code of `AccountBase` is generated in the file `src/main/generated-java/com/example/demo/domain/AccountBase.java` . Since this class is an entity, the class level annotations that were present on the original `Account` class must copied to your own `Account` class. Hopefully if you read the source code of the 'Base' class, you will find these annotations. Here is how a snapshot of the generated `AccountBase` code.

```
@MappedSuperclass
/* Copy paste these annotations as is in your subclass
@Entity
@Table(name = "ACCOUNT")
*/
public abstract class AccountBase implements Serializable {
   // code skipped for concision
}
```

As you can notice the `@MappedSuperclass` is properly generated and the other annotations are commented. Also the base class is made abstract to prevent you from instanciating it .

Note

Always check the source code of the 'Base' class, you may find some instructions related to some annotations that must be added to your class.

During the generation process, if Celerio encounters the old file

`src/main/generated-java/com/example/demo/domain/Account.java` file, it deletes it to prevent class duplication.

You can proceed by analogy to override other Java classes.

Other files (non-Java)

Code modifications through reserved folders

Ideally all generated files should be under a `generated` folder. In practice, this is not really convenient or possible as certain files are expected to be in a well known location (ex: `web.xml`)

Whenever possible and pertinent, we keep the generated by under a generated folder. This is specially useful for entity bound files as there could be a lot of them.

This strategy is currently used for localization properties files. The two folders are: `src/main/resources/localization/domain` and `src/main/resources/localization/domain-generated` . The Spring's `messageSource` configuration allow us to first lookup resources first in the domain folder and then in the domain-generated folder. This simple fact allows you to change a property key without modifying any generated file under `domain-generated` . You just have to create a properties file under the `domain` folder and 'override' your properties here. Do not forget to configure the message source to take your new file into account though.

The same applies for generated Spring Web Flow xhtml files, they are generated under `src/main/webapp/WEB-INF/flows-generated` but by configuration, Spring Web Flow will first look up a flow by id in the `src/main/webapp/WEB-INF/flows` user's reserved folder. So, do not modify the generated flows, instead simply copy the files you need under `src/main/webapp/WEB-INF/flows` and then modify the copied version.

Code modification through source control

Overwrite non-java generated file with your own content

To overwrite the content of the non-java generated file with your own content, first add add this file to your your source control system and then modify it as you want.

Next time Celerio runs, it will detect that this file is under source control and will not overwrite it.

In a sense, the generated code is frozen for this file: Celerio does not control it anymore, it is under your control.

Since you may need to have access to the orignal generated file for merging purposes, Celerio still generates it, but under the `target` folder. Please refer to the section called “Collisions and Merging” .

If you change your mind and want Celerio to generate the file again, simply remove the file from your source control and regenerate your code.

Generation rule summary for non-java file

Before generating a non-java file, Celerio applies the following rules:

Table 6.1. Generations rule summary for non-java file

File is already present on disk	File is under source control	File is the same	Celerio action
No	n/a	n/a	File is generated
Yes	Yes or No	Yes	File is not generated
Yes	Yes	No	File is generated in the collision directory
Yes	No	No	File is overwritten

Collisions and Merging

Collision folder

When Celerio detects that a file which should be generated is already present and under source control, it does not overwrite your file.

When it can, it merges the file or apply some advance rules. When it cannot resolve the conflicts Celerio generates the files in conflict in the collision folder: `target/maven-celerio-plugin/collisions`

All files generated in that folder have the same path as if they had been generated normally, except for the `target/maven-celerio-plugin/collisions` path prefix.

So, whenever appropriate, Celerio generates the original file, under the expected path, in the collision folder `target/maven-celerio-plugin/collisions`

Merging manually the files

To merge the files in conflict, you can use a tool such as WinMerge.

Since, the files that led to some conflict are located in the collision folder under the same relative path, you can use the recursive directory comparison feature.

To do so, you just have to specify in WinMerge, your project root directory and the collision directory

Then, you can keep track of changes you made and eventually merge some of the newly generated code into your existing file.

Chapter 7. Conventions and integration

Celerio has some built-in conventions. When these conventions are followed, Celerio generates cleaner Java code and some specific features. For example, by simply following some columns naming convention, you can rely on Celerio to generate all the infrastructure code and configuration that will allow you to handle file upload and download in your web application, in an optimal way.

Camel case conventions

Underscore '_' Enables Java Camel Case Syntax

'Camel Case' syntax is standard Java code convention. When Celerio encounters the character underscore '_' in a table's name or a column's name, it skips it and converts to upper case the next character when generating classes, variables or methods related to this table, or column.

Example 7.1. Basic conversion

For example, if your table name is `BOOK_COMMENT` , the generated entity class will be named `BookComment` ; a variable holding `BookComment` instance will be named `bookComment` and a setter will be named `setBookComment` , etc.

Native camel case support

If your table's and/or column's name use a camelCase syntax AND if the JDBC driver preserves this syntax, then Celerio takes it into account when generating classes, variables or methods related to this table, or column.

Example 7.2. Example

For example, if your table name is `bankAccount` , the generated entity class will be named `BankAccount` ; a variable holding `BankAccount` instance will be named `bankAccount` and a setter will be named `setBankAccount` , etc.

Choosing explicit names for your tables and columns is thus very important as it improves your source code readability without the burden of relying on configuration.

Primary key conventions

Numerical Primary Keys

Each numerical primary key column is mapped with `@GeneratedValue` and `@Id` annotations.

> **Important**
>
> If your database does not support identity columns, you should create the sequence 'hibernate_sequence'. Please refer to Hibernate reference documentation for more advanced alternatives.

Primary Keys with 32 characters

By convention, for all primary keys that are char(32), Celerio maps the column with the following annotations

```
@GeneratedValue(generator = "strategy-uuid")
@GenericGenerator(name = "strategy-uuid", strategy = "uuid")
@Id
```

annotations. so it uses Hibernate's UUIDHexGenerator. Therefore no sequence is needed for these primary keys.

Other Primary Keys

For primary key that are char(x) where x is different from 32, Celerio map the column with an "assigned" generator, which means you must provide manually the primary key value.

The 'Account' table

The Account table is a special table that contains the user's login and password columns and eventually the email and enabled columns. It has an important role during the login phase. It is also used by the `AccountContext` generated class which store the current `account` information in the current thread.

Celerio detects automatically your 'Account' table. An account table candidate is expected to have at least the following columns:

Table 7.1. Account table conditions

Column's name	Mapped Java Type	Description
"username" OR "login" OR "user_name" OR "identifiant"	String	Login used by the end user to authenticate to this web application
"password" OR "pwd" OR "passwd" OR "mot_de_passe" OR	String	Password (in clear) used by the end user to authenticate to this

Column's name	Mapped Java Type	Description
"motdepasse"		web application

If no Account table candidate is found, Celerio will do as if it had found one and will generate a mock Account DAO implementation that returns 2 dummy users (user/user and admin/admin) instead of generating a real JPA DAO implementation. It is up to you to replace this DAO implementation with your own implementation.

Note

You may also elect the account table by configuration.

The 'Role' table

The Role table is a special table that contains the account's roles. To be detected by Celerio, it must have a many-to-many or a many-to-one relationship with the found 'Account' table and contain the following mandatory column:

Table 7.2. Role table conditions

Column's name	Mapped Java Type	Description
"authority" OR "role_name" OR "role" OR "name_locale"	String	The generated code relies on the following authority's values: `ROLE_USER`, `ROLE_ADMIN`

Here is a sample SQL script (H2 Database) that complies to Celerio conventions

```
CREATE TABLE ACCOUNT (
   account_id char(32) not null, login
   varchar(255) not null,
   password varchar(255) not null,
   email varchar(255) not null,

   constraint account_unique_1 unique (login),
   constraint account_unique_2 unique (email),
   primary key (account_id)
);
CREATE TABLE ROLE (
   role_id smallint generated by default as identity,
   name_locale varchar(255) not null,

   constraint role_unique_1 unique (name_locale),
   primary key (role_id)
);
CREATE TABLE ACCOUNT_ROLE (
```

```
    account_id char(32) not null,
    role_id smallint not null,

    constraint account_role_fk_1 foreign key (account_id) references ACCOUNT,
    constraint account_role_fk_2 foreign key (role_id) references ROLE,
    primary key (account_id, role_id)
);
```

Other optional account's columns

The Email column

If the detected Account table has an email column, it is used by the generated code in few places, mostly as an illustration of the EmailService usage.

Table 7.3. Account's table email column conditions

Column's name	Mapped Java Type	Description
"email", "email_address", "email-Address", "mail"	String	The user's email.

The Enabled column

If the detected Account table has an enabled column, it is used by the generated code related to Spring Security integration.

Table 7.4. Account's table enabled column conditions

Column's name	Mapped Java Type	Description
"enabled" OR "is_enabled" OR "isenabled"	Boolean	Only enabled users (enabled == true) can login.

Special columns for file handling support

When the following columns are present simultaneously in a table, Celerio generates various helper methods to ease file manipulation from the web tier to the persistence layer.

- 'prefix'_FILE_NAME (String)

- 'prefix'_CONTENT_TYPE (String)
- 'prefix_SIZE or 'prefix'_LENGTH (int)
- 'prefix'_BINARY (blob)

Example 7.3. Example of binary

```
mydoc_content_type      varchar(255)    not null,
mydoc_size              integer         not null,
mydoc_file_name         varchar(255)    not null,
mydoc_binary            bytea,
```

This convention will allow you to upload a file transparently, save it to the corresponding table, then download it using a simple URL.

ACCOUNT_ID column & Hibernate Filter

When a table contains a foreign key pointing to the Account table, Celerio assumes that the content of this table belongs to the user represented by the account_id foreign key.

An hibernate filter is configured to make sure that this table is loaded only by the current user.

The filter is enabled by the `HibernateFilterInterceptor`. Of course this default behavior may not always suit your needs. There are two ways of disabling it:

1. Remove the `@Filter` annotation from the Entity. This imply taking control over the entity.
2. Disable the filter programmatically using the HibernateFilterContext generated helper.
3. Disable globally this convention in Celerio's configuration file.

Version column and Optimistic Locking

If your table has a column named `VERSION` whose type is an int, Celerio assumes by convention that you want to enable an optimistic locking strategy. As a result, the property is annotated with `@Version` .

Many to many and inverse attribute

Which side of the relation is marked as inverse="true" ? By convention, the side whose corresponding

column's order is the highest on the "Middle table".

Chapter 8. Configuration File

Main Configuration File

The main Celerio configuration file is by convention located under `${basedir}/src/main/config/maven-celerio-plugin/maven-celerio-plugin.xml` The bootstrap goal creates one that you can customize. This configuration file must respect the `celerio.xsd` schema.

Please refer to Chapter 12, *XSD defaults* for a complete XSD reference.

Splitting the `entityConfigs` tag

The `entityConfigs` tag can be declared in the main Celerio configuration file AND/OR in sub-configuration files that are included in the main configuration file. Note that, for such included files, only the `entityConfigs` tag is loaded by Celerio. Here is an example:

```
<celerio>
 <includes>
  <include filename="maven-celerio-plugin-secondary.xml"/>
 </includes>
 <entityConfigs>
  <entityConfig tableName="BOOK" subPackage="fromMainConfig"/>
 </entityConfigs>
</celerio>
```

And here is the included file

```
<celerio>
 <entityConfigs>
  <entityConfig tableName="ACCOUNT" subPackage="fromIncludeConfig"/>
 </entityConfigs>
</celerio>
```

Template pack configuration file

The template packs can be declared in the main Celerio configuration file OR in the file `${basedir}/src/main/config/maven-celerio-plugin/celerio-template-packs.xml` . When this file is present Celerio loads from it the `packs` tag instead of loading it from the main Celerio configuration file.

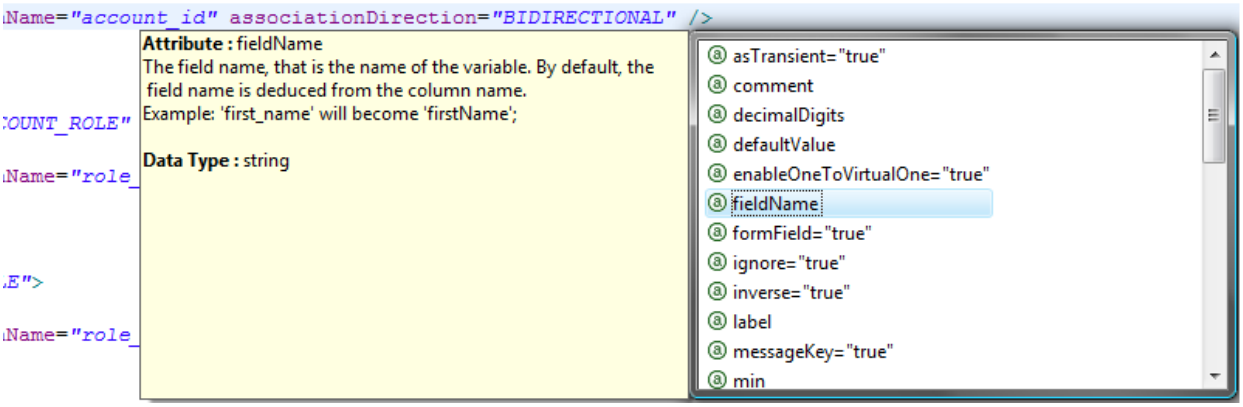
```
<celerio>
 <configuration>
  <packs>
```

```
  <pack name="pack-backend" enable="true">
   <filenames><!-- do not generate these files -->
    <filename include="false" pattern="src/main/resources/database.properties" />
    <filename include="false" pattern="src/main/resources/hibernate.properties" />
    <filename include="false" pattern="src/main/resources/log4j.properties" />
    <filename include="false" pattern="src/main/resources/ehcache/local.xml" />
   </filenames>
  </pack>
 </packs>
</configuration>
</celerio>
```

This separated file was mainly introduced for multi-module maven projects. The various modules (e.g. core and web) shares the same `maven-celerio-plugin.xml` configuration file but uses their own `celerio-template-packs.xml` configuration file.

Chapter 9. Configuration

Before editing your configuration file, make sure that Eclipse displays the documentation present in the `celerio.xsd` file and that it suggests the available tags. From Eclipse, you cannot work efficiently without the help of the XSD documentation.



Tag completion and documentation under Eclipse

Id

If you rely on conventions, you do not need to configure anything regarding Ids. These examples are for advanced usage.

Use a SEQUENCE per TABLE

In case you use a sequence per table to generate your primary key values, you must configure Celerio in order to take it into account. Here is an example:

```
<entityConfigs>
  <entityConfig tableName="ADDRESS" sequenceName="ADDRESS_SEQ"/>
</entityConfigs>
```

assuming the PK of the ADDRESS table is mapped to an Integer, here is how would look the generated code:

```
@Column(name = "ADDRESS_ID", nullable = false, unique = true, precision = 5)
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "ADDRESS_SEQ")
@Id
@SequenceGenerator(name = "ADDRESS_SEQ", sequenceName = "ADDRESS_SEQ")
public Integer getAddressId() {
  return addressId;
}
```

Use a custom Id generator

In certain cases, generally when you work with legacy databases, you may need to use a custom Id generator in order to be consistent with the legacy system. Here is an example:

```
<entityConfig tableName="ADDRESS">
 <columnConfigs>
  <columnConfig columnName="ADDRESS_ID">
   <generatedValue generator="myCustomerGenerator" />
   <genericGenerator name="myCustomerGenerator" strategy="com.yourcompany.appli.customgen.CustomerGenerator">
    <parameters>
     <parameter name="sequence" value="YOUR_EVNTUAL_SEQ" />
    </parameters>
   </genericGenerator>
  </columnConfig>
 </columnConfigs>
</entityConfig>
```

leads to:

```
@Column(name = "ADDRESS_ID", nullable = false, unique = true, precision = 5)
@GeneratedValue(generator = "myCustomerGenerator")
@GenericGenerator(name = "myCustomerGenerator",
 strategy = "com.yourcompany.appli.customgen.CustomerGenerator",
 parameters = @Parameter(name = "sequence", value = "YOUR_EVNTUAL_SEQ"))
@Id
public Integer getAddressId() {
  return addressId;
}
```

Entity and property names

Force an entity name

By default, an entity name is deduced from the Table name. To force the entity name to a different value, use the `entityName` attribute of the `entityConfig` element, for example.

```
<entityConfigs>
 <entityConfig tableName="ACCOUNT" entityName="UserAccount"/>
</entityConfigs>
```

Force a property name

By default, a property name is deduced from the column name. To force the property name to a different value, use the `fieldName` attribute of the `columnConfig` element, for example.

```
<entityConfigs>
 <entityConfig tableName="ACCOUNT">
  <columnConfigs>
   <columnConfig columnName="user_dob" fieldName="birthDate"/>
  </columnConfigs>
```

```
  </entityConfig>
</entityConfigs>
```

leads to:

```
Date birthDate;
```

Advanced property name calculation

By default Celerio calculates Java field name based on the underlying column name. The `fieldNaming` element allows you to change the column names passed to Celerio to calculate the default field names. The example below removes well known prefix pattern from column names:

```
<configuration>
  <conventions>
    <fieldNaming regexp="^.{3}_{1}" replace=""/>
  </conventions>
</configuration>
```

In that case, column names such as `XYZ_SOMETHING_MEANINGFUL` now lead to Java field name `sometingMeaningful` instead of `xyzSometingMeaningful`.

Type Mapping

Celerio has some conventions regarding type mapping. You can change them either locally or globally using rules.

Force a type mapping locally

You can force the mapped type using the `mappedType` attribute of the columnConfig element. For example to force a mapping to an Integer you would do:

```
<entityConfigs>
  <entityConfig tableName="ACCOUNT">
    <columnConfigs>
      <columnConfig columnName="year" mappedType="M_INTEGER"/>
    </columnConfigs>
  </entityConfig>
</entityConfigs>
```

Number mapping customization

You can configure number mapping rules globally. For example, to map all columns whose size and decimal digits are > 1 to BigDecimal, proceed as follow:

```
<configuration>
  <numberMappings>
```

```
   <numberMapping mappedType="M_BIGDECIMAL" columnSizeMin="1" columnDecimalDigitsMin="1"/>
  </numberMappings>
</configuration>
```

First rule that matches is used. For example to map to either Boolean, Double or BigDecimal you can do:

```
<configuration>
  <numberMappings>
   <numberMapping mappedType="M_BOOLEAN" columnSizeMin="1" columnSizeMax="2" columnDecimalDigitsMin="0"
columnDecimalDigitsMax="1"/>
   <numberMapping mappedType="M_DOUBLE" columnSizeMin="1" columnSizeMax="11" columnDecimalDigitsMin="1"
columnDecimalDigitsMax="4"/>
   <numberMapping mappedType="M_BIGDECIMAL" columnSizeMin="11" columnDecimalDigitsMin="4"/>
  </numberMappings>
</configuration>
```

Note that the `columnSizeMin` is inclusive and `columnSizeMax` is exclusive.

Date mapping customization

You can configure date mapping rules globally. For example, to map all date/time/timestamp column to Joda Time LocalDateTime, proceed as follow:

```
<configuration>
  <dateMappings>
   <dateMapping mappedType="M_JODA_LOCALDATETIME" />
  </dateMappings>
</configuration>
```

And for example to map differently the columns whose name is VERSION you can add the following mapping rule:

```
<configuration>
  <dateMappings>
   <dateMapping mappedType="M_UTILDATE" columnNameRegExp="VERSION"/>
   <dateMapping mappedType="M_JODA_LOCALDATETIME" />
  </dateMappings>
</configuration>
```

Associations

@ManyToOne

By default, Celerio generates the code for a `@ManyToOne` association when it encounters a column having a `foreign key` constraint and no `unique` constraint.

The variable name of the many to one association is deduced by default from the `fieldName` of the column. For example if the `fieldName` is `addressId` , the many to one variable name will be `address` . In case where the `fieldName` already matches the name of the target entity, Celerio adds the

"Ref" suffix to the variable name. Here are few simplified examples:

```
// column name is 'addr_id'
@ManyToOne Address addr;

// column name is 'address'
@ManyToOne Address addressRef;

// column name is 'anything_else'
@ManyToOne Address address;
```

In any case, use the `manyToOneConfig` element to force a different variable name. For example:

```
<columnConfig columnName="addr_id">
   <manyToOneConfig var="myAddress"/>
</columnConfig>
```

will lead to

```
@ManyToOne Address myAddress;
```

The `manyToOneConfig` element also allows you to tune the JPA fetch type and the JPA cascade types. Please refer to the XSD for more information.

If you have some inheritance involved on the 'one' side of the many to one association, the table referenced by the foreign key is not enough to identify the target entity. In that case, set the `targetEntityName` attribute of the `columnConfig` element. For example:

```
<columnConfig columnName="address_id" targetEntityName="HomeAddress"/>
```

On legacy schema, the foreign key constraint may not be present and Celerio will not generate the many to one association you would expect. Hopefully you can configure Celerio to do as if a foreign key constraint was present by setting the `targetTableName` attribute of the `columnConfig` element. For example:

```
<columnConfig columnName="address_id" targetTableName="ADDRESS"/>
```

@OneToMany

One to many association is configured on the 'many' side of the association, more precisely on the same `columnConfig` as the one used for the associated many to one association. This may be a bit confusing at first but it has the advantage to group together, both associations on the side that really owns the association.

Celerio generates the code for one to many association when a many to one association is present and when the `associationDirection` attribute of the `columnConfig` element is `BIDIRECTIONAL`. For example:

```
<entityConfig tableName="Account">
   <columnConfig columnName="address_id"
            associationDirection="BIDIRECTIONAL"/>
</entityConfig>
```

will lead (assuming address_id refers to Address) to something like:

```
// in Account.java
@ManyToOne Address address;
```

```
// In Address.java
@OneToMany List<Account> accounts;
```

In the example above `accounts` is simply the plural of the Account entity that Celerio guessed. We were of course lucky on this one.

Use the `oneToManyConfig` element of the `columnAttribute` to set the name of the one to many association to a different value. As you will see, you can also set the name of an element of the collection to control the name of the associated helper methods that Celerio generates (adder, remover, etc.). Here is an example:

```
<entityConfig tableName="Account">
   <columnConfig columnName="address_id"
            associationDirection="BIDIRECTIONAL">
      <oneToManyConfig var="people" elementVar="resident"/>
   </columnConfig>
</entityConfig>
```

will lead to

```
// In Address.java
@OneToMany List<Account> people;

public void addResident(Account resident) {
// skip...
}
```

The `oneToManyConfig` element also allows you to tune the JPA fetch type and the JPA cascade types. Please refer to the XSD for more information.

@OneToOne

By default, Celerio generates the code for a `@OneToOne` association when it encounters a column having a `foreign key` constraint AND a `unique` constraint.

One to one associations are very similar to many to one associations. To change the variable name, the JPA fetch type or the cascade types of the one to one association, use the `oneToOne` element of the `columnConfig` element.

Inverse @OneToOne

Inverse one to one association is for one to one association what one to many association is for many to one association.

Celerio generates the code for inverse one to one association when a one to one association is present and when the `associationDirection` attribute of the `columnConfig` element is `BIDIRECTIONAL`.

Inverse one to one association is configured on the owning side of association, that is on the `columnConfig` that has the foreign key and unique constraints. As for one to many association, this may be a bit confusing at first but it has the advantage to group together, both associations on the side that really owns the association.

@ManyToMany

Many to many association necessarily involves a join table. When Celerio detects a join table, it generates the code for the many to many relation. Celerio assumes that a table is a join table when it has 2 foreign keys and no other columns, except eventually a primary key column and a column used for optimistic locking.

To fine tune the many to many association, you must declare an entityConfig for the join table. You may use the `manyToManyConfig` element. to set the related variables and adder/remover/etc. method names. You can use the `inverse` attribute to force the inverse side of the association. For example:

```
<entityConfig tableName="account_role" associationDirection="BIDIRECTIONAL">
 <columnConfigs>
  <columnConfig columnName="account_id">
   <manyToManyConfig var="theAccounts" elementVar="anAccount"/>
  </columnConfig>
  <columnConfig columnName="role_id" inverse="true">
   <manyToManyConfig var="theRoles" elementVar="aRole"/>
  </columnConfig>
 </columnConfigs>
</entityConfig>
```

Note

In case Celerio does not detect the join table, for example if an extra column is present, you can force it by setting to `true` the `middleTable` attribute of the `entityConfig` element.

Chapter 10. Best practices

Source control

Celerio works in harmony with a source-control system (SVN or CVS). To really take advantage of Celerio's features, you must use a source-control system for the your project. However, it is important to commit only what is strictly necessary as we will see.

When the project starts

As soon as you start your project you should commit in your source control at least the files that are generated by the `bootstrap` goal, that is, the `pom.xml` file, the Celerio configuration file and the SQL files. This way Celerio won't overwrite them each time you regenerate your project.

Certain generated files are meant to be modified to fit your business needs. You should not hesitate to commit them early in your development process. In general this is the case for the following files: `src/main/resources/spring/applicationContext-http-security.xml`, `src/main/resources/spring/springmvc-webapp.xml`, etc.

Do not commit the generated Java files

Never commit the folders `src/main/generated-java` or `src/test/generated-java` . You would have keep them in sync manually with your source control each time you regenerate your project. As for classes generated by the compiler, you should not commit the Java generated files.

Pages or Controllers

We recommend to do not commit the generated JSP/xhtml pages and to do not take control over the generated Java controllers (`<TableName>Controller.java`)

Instead, create your own JSP and controllers and use the generated ones as source for handy copy paste.

Indeed, there is absolutely no reason that the generated JSPs will fit the end user need. This part is really close to the application business. Keep the generated page as a reference or eventually a simple administration zone of your application.

Use a 'Development' Database

At least during the early phase of your development, we encourage you to use a database such as H2Database to be more productive. Indeed, there is nothing to install or configure and you can experiment easily some new database schema.

Chapter 11. Troubleshootings

Generated code does not compile

You may encounter a situation where the generated Java code does not compile. This could be due to a Celerio's bug. To avoid loosing too much time on this issue, if you know how to fix the compilation error, you should take totally control over the generated file that does not compile (please refer to Chapter 6, *Code modification, regeneration and collisions*) and fix the compilation error manually.

Make sure you also report the issue to `<support@jaxio.com>` so it get fixed in a next release of Celerio. As soon as the issue is fixed in Celerio, you may give the control back to Celerio by simply removing the file so Celerio can generate it as usual.

Tomcat converts request parameters to 0, "" or true.

By default Tomcat coerces null values of types such as `java.lang.String` or `java.lang.Boolean` to `""` or `false`.

Please edit your `catalina.properties` file and add the following property:

```
org.apache.el.parser.COERCE_TO_ZERO=false
```

Please refer to Tomcat System properties for more information.

Chapter 12. XSD defaults

Simple types

`MethodConvention`

Table 12.1. `MethodConvention` default parameters

Name	Documentation	prefix	suffix
GET		get	
SET		set	
ADD		add	
EDIT		edit	
CONTAINS		contains	
GET_BY		getBy	
DELETE_BY		deleteBy	
REMOVE		remove	
REMOVE_ALL		removeAll	
HAS		is	Set
GET_LOCALIZED		get	Localized
RANDOM_GETTER		get	Random

`EnumType`

Table 12.2. `EnumType` default parameters

Name	Documentation
ORDINAL	Persist enumerated type property or field as an integer
STRING	Persist enumerated type property or field as a string
CUSTOM	Persisted via a custom user type

Module

Table 12.3. `Module` default parameters

Name	Documentation
SPRING3	
PACK_MVC_3	
JAVAX_VALIDATION	
PRODUCE_HAS_METHODS	
ENABLE_FK_COLUMN_SETTER	
PRODUCE_TO_DISPLAY_STRING_METHOD	
COPYABLE	
CHAR_PADDING	
PRIMEFACES	
PRIMEFACES_FILE_HANDLER	

ClassType

Table 12.4. `ClassType` default parameters

Name	Documentation	prefix	suffix	subPackage	generatedPackage
model			GeneratedPackage.Model		
primaryKey		Pk	GeneratedPackage.Model		
dao		Dao	GeneratedPackage.Dao		
formatter		Formatter	GeneratedPackage.Formatter		
hibernate		DaoImpl	GeneratedPackage.Hibernate		
manager		Service	GeneratedPack-		

Name	Documentation	prefix	suffix	subPackage	generatedPackage
			age.Manager		
managerImpl		ServiceImpl	GeneratedPackage.ManagerImpl		
validator		Validator	GeneratedPackage.WebModel Validator		
memory		Memory	GeneratedPackage.Memory		
enumModel			GeneratedPackage.EnumModel		
enumItems		Items	GeneratedPackage.EnumItems		
modelGenerator		Generator	GeneratedPackage.Manager		
controller		Controller	GeneratedPackage.WebController		
controllerWithPathVariable		ControllerWithPathVariable	GeneratedPackage.WebController		
restController		RestController	GeneratedPackage.RestController		
entityForm		Form	GeneratedPackage.WebModel EntityForm		
searchForm		SearchForm	GeneratedPackage.WebModel SearchForm		
formService		FormService	GeneratedPackage.WebController		
formValidator		FormValidator	GeneratedPackage.WebController		
searchCon-		SearchControl-	GeneratedPack-		

Name	Documentation	prefix	suffix	subPackage	generatedPackage
troller		ler	age.WebController		
webSupport		WebSupport	GeneratedPackage.WebController		
webModel			GeneratedPackage.WebModel		
webModelConverter		Converter	GeneratedPackage.WebModelConverter		
webConverter		Converter	GeneratedPackage.WebConverter		
webModelItems		Items	GeneratedPackage.WebModelItems		
wicket			GeneratedPackage.Wicket		

CascadeType

Defines the set of cascadable operations that are propagated to the associated entity. The value <code>cascade=ALL<code> is equivalent to <code>cascade={PERSIST, MERGE, REMOVE, REFRESH}</code>. @since Java Persistence 1.0

Table 12.5. CascadeType default parameters

Name	Documentation
ALL	Cascade all operations
PERSIST	Cascade persist operation
MERGE	Cascade merge operation
REMOVE	Cascade remove operation
REFRESH	Cascade refresh operation

GenerationType

Defines the types of primary key generation. @since Java Persistence 1.0

Table 12.6. `GenerationType` default parameters

Name	Documentation
TABLE	Indicates that the persistence provider must assign primary keys for the entity using an underlying database table to ensure uniqueness.
SEQUENCE	Indicates that the persistence provider must assign primary keys for the entity using database sequence column.
IDENTITY	Indicates that the persistence provider must assign primary keys for the entity using database identity column.
AUTO	Indicates that the persistence provider should pick an appropriate strategy for the particular database. The <code>AUTO</code> generation strategy may expect a database resource to exist, or it may attempt to create one. A vendor may provide documentation on how to create such resources in the event that it does not support schema generation or cannot create the schema resource at runtime.

`TableType`

Table 12.7. `TableType` default parameters

Name	Documentation
TABLE	
VIEW	
ALIAS	not supported
SYNONYM	not supported

`WellKnownFolder`

Table 12.8. `WellKnownFolder` default parameters

Name	Documentation	folder	generatedFolder
JAVA		src/main/java	src/main/generated-java
JAVA_TEST		src/test/java	src/test/generated-java
WEBAPP		src/main/webapp	
WEBINF		src/ main/webapp/WEB-INF	
VIEWS		src/ main/ webapp/ WEB-INF/views	
FLOWS		src/ main/ webapp/WEB-INF/flows	src/ main/ webapp/ WEB-INF/flows-generated
RESOURCES		src/main/resources	src/main/resources
RESOURCES_TEST		src/test/resources	src/test/resources
LOCALIZATION		src/ main/re-sources/localization	
DO-MAIN_LOCALIZATIO N		src/main/resources/" + LOCALIZA-TION.getResourcePath() + "/" + Mod-el.getSubPackagePath() + "-generated	
SPRING		src/ main/resources/spring	
SPRING_TEST		src/test/resources/spring	
CEL-ERIO_LOCAL_TEMPL ATE		src/main/celerio/	
COLLISION		target/ maven-celerio-plugin/	
SQL		src/main/sql	
CONFIG		src/main/config	

Name	Documentation	folder	generatedFolder
SITE		src/site/	

TrueFalse

Table 12.9. TrueFalse default parameters

Name	Documentation
TRUE	
FALSE	

InheritanceType

Defines inheritance strategy options. @since Java Persistence 1.0

Table 12.10. InheritanceType default parameters

Name	Documentation
SINGLE_TABLE	A single table per class hierarchy
TABLE_PER_CLASS	A table per concrete entity class
JOINED	A strategy in which fields that are specific to a sub-class are mapped to a separate table than the fields that are common to the parent class, and a join is performed to instantiate the subclass.

JdbcType

Table 12.11. JdbcType default parameters

Name	Documentation	logger	jdbcType
ARRAY	Not supported	Types.ARRAY	
BIGINT		Types.BIGINT	

Name	Documentation	logger	jdbcType
BINARY		Types.BINARY	
BIT		Types.BIT	
BLOB		Types.BLOB	
BOOLEAN		Types.BOOLEAN	
CHAR		Types.CHAR	
CLOB		Types.CLOB	
DATALINK	Not supported	Types.DATALINK	
DATE		Types.DATE	
DECIMAL		Types.DECIMAL	
DISTINCT	Not supported	Types.DISTINCT	
DOUBLE		Types.DOUBLE	
FLOAT		Types.FLOAT	
INTEGER		Types.INTEGER	
JAVA_OBJECT		Types.JAVA_OBJECT	
LONGVARBINARY		Types.LONGVARBINARY	
LONGVARCHAR		Types.LONGVARCHAR	
NUMERIC		Types.NUMERIC	
OTHER	Not supported	Types.OTHER	
REAL		Types.REAL	
REF		Types.REF	
SMALLINT		Types.SMALLINT	
STRUCT	Not supported	Types.STRUCT	
TIME		Types.TIME	
TIMESTAMP		Types.TIMESTAMP	
TINYINT		Types.TINYINT	
VARBINARY		Types.VARBINARY	
VARCHAR		Types.VARCHAR	
ROW_ID		Types.ROWID	
LONGNVARCHAR		Types.LONGNVARCHAR	
NCHAR		Types.NCHAR	
NCLOB		Types.NCLOB	

Name	Documentation	logger	jdbcType
NVARCHAR		Types.NVARCHAR	
NULL	Not supported	Types.NULL	
SQLXML		Types.SQLXML	

GeneratedPackage

Table 12.12. GeneratedPackage default parameters

Name	Documentation	subPackage	rootPackage
AccountService		service.account	
Model		domain	
Context		context	
Dao		dao	
DaoSupport		dao.support	
Validation		validation	
ValidationImpl		validation.impl	
EmailService		service.email	
Hibernate		dao	
HibernateListener		dao.hibernate.listener	
HibernateSupport		dao.hibernate	
Jms		jms	
Jmx		jmx	
Jwebunit		jwebunit	
Manager		service	
ManagerImpl		service	
ManagerSupport		service.support	
Memory		memory	
PasswordService		service.password	
Random		random	
ReminderService		service.reminder	
Root			

Name	Documentation	subPackage	rootPackage
Scheduling		scheduling	
Security		security	
Service		service	
SignupService		service.signup	
Transaction		transaction	
Util		util	
Web		web	
WebAction		web.action	
WebContext		web.context	
WebController		web.controller	
RestController		web.controller	
WebModel		web.domain	
WebModelValidator		web.domain	
WebModelSupport		web.domain.support	
WebModelConverter		web.domain	
Formatter		formatter	
FormatterSupport		formatter.support	
WebComponent		web.component	
WebConverter		web.converter	
WebModelItems		web.domain	
WebModelEntityForm		web.domain	
WebModelSearchForm		web.domain	
WebFaces		web.faces	
WebFlow		web.flow	
WebFilter		web.filter	
WebInterceptor		web.interceptor	
WebListener		web.listener	
WebServlet		web.servlet	
WebUtil		web.util	
WebValidator		web.validator	
WebUi		web.ui	

Name	Documentation	subPackage	rootPackage
WebEl		web.ui.el	
GwtClient		web.client	
GwtShared		web.shared	
GwtServer		web.server	
Wicket		web.wicket	
WicketComponent		web.wicket.component	
WicketComponent-Form		web.wicket.component.form	
WicketListener		web.wicket.listener	
WicketPage		web.wicket.page	
WicketPanel		web.wicket.panel	
WicketSkin		web.wicket.skin	
WicketUtil		web.wicket.util	
EnumModel		domain	
EnumItems		web.domain	
Converter		converter	

AssociationDirection

Table 12.13. AssociationDirection default parameters

Name	Documentation
UNIDIRECTIONAL	
BIDIRECTIONAL	

FetchType

Defines strategies for fetching data from the database. The <code>EAGER</code> strategy is a requirement on the persistence provider runtime that data must be eagerly fetched. The <code>LAZY</code> strategy is a hint to the persistence provider runtime that data should be fetched lazily when it is first accessed. The implementation is permitted to eagerly fetch data for which the <code>LAZY</code> strategy hint has been specified. In particular, lazy fetching might only be available for {@link Basic} mappings for which property-based access is used. <pre> Example: @Basic(fetch=LAZY) protected String getName() { return name; } </pre> @since Java Persistence 1.0

Table 12.14. `FetchType` default parameters

Name	Documentation
LAZY	Defines that data can be lazily fetched
EAGER	Defines that data must be eagerly fetched

`MappedType`

Table 12.15. `MappedType` default parameters

Name	Documentation	javaType	fullJavaType	isNow
M_ARRAY		Array	java.sql.Array	
M_BIGDECIMAL		BigDecimal	java.math.BigDecimal	
M_BIGINTEGER		BigInteger	java.math.BigInteger	
M_BOOLEAN		Boolean	java.lang.Boolean	
M_BYTES		byte[]	byte[]	
M_CLOB		String	java.lang.String	
M_DOUBLE		Double	java.lang.Double	
M_FLOAT		Float	java.lang.Float	
M_BLOB		byte[]	byte[]	
M_INTEGER		Integer	java.lang.Integer	
M_LONG		Long	java.lang.Long	
M_REF		Ref	java.sql.Ref	
M_STRING		String	java.lang.String	
M_CHAR		Character	java.lang.Character	
M_BYTE		Byte	java.lang.Byte	
M_JODA_LOCALDATE		LocalDate	org.joda.time.LocalDate	
M_JODA_LOCALDATETIME		LocalDateTime	org.joda.time.LocalDateTime	
M_SQLDATE		java.sql.Date	java.sql.Date	

Name	Documentation	javaType	fullJavaType	isNow
M_UTILDATE		Date	java.util.Date	
M_TIME		java.sql.Time	java.sql.Time	
M_TIMESTAMP		java.sql.Timestamp	java.sql.Timestamp	
M_URL		java.net.URL	java.net.URL	
M_OBJECT		Object	java.lang.Object	

CollectionType

Table 12.16. CollectionType default parameters

Name	Documentation	fullType	fullImplementation
ArrayList		java.util.List	java.util.ArrayList
HashSet		java.util.Set	java.util.HashSet

Complex types

metaAttribute

Meta attributes are free form key value pairs

Table 12.17. metaAttribute properties

Name	Type	Documentation
name	string	name of you meta attribute
value	string	value for this attribute

conventions

Change the default celerio conventions to your own needs.

Table 12.18. conventions properties

Name	Type	Documentation
fieldNaming	fieldNaming	Allows you to change the way Celerio calculates the default field name out of a column name.
eclipseFormatter	eclipseFormatter	Defines the formatting option of the generated Java files.
xmlFormatter	xmlFormatter	Defines the formatting options of the generated XML/XHTML files.
classTypes	classTypeOverride []	Override the conventions for classes
generatedPackages	generatedPackageOverride []	Override the conventions for packages
methodConventions	methodConventionOverride []	Override the conventions for methods
wellKnownFolders	wellKnownFolderOverride []	Override the conventions for folders
collectionType	collectionType	You can override the default collection type for this entity
identifiableProperty	string	The property name used in the Identifiable interface. Defaults to 'primaryKey'. If all your primary key are mapped to the same property name, you should change the identifiable property here to limit redundancy.
entitySubPackage-Preprended	trueFalse	When constructing the package name of a class constructed using a GeneratedPackage, tell if the GeneratedPackage subPackage should be appended. For example given the entity 'MyEntity' with subpackage 'mysubpackage', and the generated package Manager-Impl with subpackage 'impl' then the packageName of all classes for MyEntity constructed using ManagerImpl will have the subpackage 'impl.mysubpackage'

databaseInfo

Information about the database where celerio extracted the metadata

Table 12.19. databaseInfo properties

Name	Type	Documentation
databaseMajorVersion	int	
databaseMinorVersion	int	
databaseProductName	string	
databaseProductVersion	string	
driverMajorVersion	int	
driverMinorVersion	int	
driverName	string	
driverVersion	string	
extraInfo	string	

wellKnownFolderOverride

change the convention for a given well known folder

Table 12.20. wellKnownFolderOverride properties

Name	Type	Documentation
wellKnownFolder	wellKnownFolder	WellKnownFolder to override
folder	string	Override the folder for this WellKnownFolder
generatedFolder	string	Override the generated folder for this WellKnownFolder

enumValue

Table 12.21. enumValue properties

Name	Type	Documentation
comments	string []	Set comments for this enum value.
value	string	Value Example: MS
name	string	Name of the enum value, by default is is the one defined in value Example: Miss
label	string	Label to be used when representing this enum value Example: gender.male

implementsInterface

Table 12.22. implementsInterface properties

Name	Type	Documentation
fullType	string	The full interface name that this entity implements. For example 'com.mycompany.MyInterface'

customAnnotation

Table 12.23. customAnnotation properties

Name	Type	Documentation
annotation	string	The full qualified custom annotation to apply to this property. For example: @com.mycompany.MyAnnotation(debug = true)

pack

A pack is the aggregation of templates and static files that produces functionalities.

Table 12.24. `pack` properties

Name	Type	Documentation
`filenames`	pattern []	Control the generation output by filtering the generated files based on their filename.
`templates`	pattern []	Control the generation output by filtering the execution of the generation templates based on their filename.
`name`	string	Name of the pack
`path`	string	Path of the pack, it should be relative to the project, or absolute. Example: src/main/packs/my-own-pack/
`enable`	boolean	Should this pack be used ?
`order`	int	Specify the pack order, its main interest is when two packs produce the same artifacts.

`manyToOneConfig`

Table 12.25. `manyToOneConfig` properties

Name	Type	Documentation
`cascades`	cascade []	The list of JPA cascade types for the this ManyToOne association.
`var`	string	The variable name for this many-to-one relation. It should be singular, for example: 'parent'.
`fetch`	fetchType	The JPA fetch type for this ManyToOne association.
`ajax`	boolean	Should this many to one be represented as an ajax drop down instead of a simple list ?

index

Description of the given table's indices and statistics

Table 12.26. index properties

Name	Type	Documentation
columnName	string	Column name
indexName	string	Index name
nonUnique	boolean	Can index values be non-unique

generatedPackageOverride

Override the convention for a given GeneratedPackage

Table 12.27. generatedPackageOverride properties

Name	Type	Documentation
generatedPackage	generatedPackage	The GeneratedPackage to override
rootPackage	string	Override the root package Example: com.yourcompany
subPackage	string	Override the sub package, if rootPackage is also specified they will be merged. Example: my.subpackage

restriction

Table 12.28. restriction properties

Name	Type	Documentation
classTypes	classType []	Restrict the generation to the following classTypes

Name	Type	Documentation
wellKnownFolders	wellKnownFolder []	Restrict the generation to the following wellKnownFolders
generatedPackages	generatedPackage []	Restrict the generation to the following generatedPackages

inheritance

Table 12.29. inheritance properties

Name	Type	Documentation
discriminatorColumn	string	
discriminatorValue	string	
parentEntityName	string	
strategy	inheritanceType	

fieldNaming

By default Celerio calculates Java field name based on the underlying column name. This setting allows you to change the column name that is passed to Celerio to calculate the default field name. You can for example remove well known prefix pattern from your column names.

Table 12.30. fieldNaming properties

Name	Type	Documentation
regexp	string	The regular expression to apply on the column name. For example, assuming you want to remove from all column names the prefix string that consists of 3 chars and a '_', you can use 'regexp="^.{3}_{1}" replace=""'
replace	string	The replacement String. For example, assuming you want to remove from all column names the prefix string that consists of 3

Name	Type	Documentation
		chars and a '_', you can use 'regexp="^.{3}_{1}" replace="".

oneToManyConfig

Table 12.31. oneToManyConfig properties

Name	Type	Documentation
cascades	cascade []	The list of JPA cascade types for the this OneToMany association.
var	string	The variable name for the collection. It should be plural, for example: 'children'.
elementVar	string	The variable name for an element of the collection. For example, if the variable name for the collection is 'children', the elementVar should be child. This elementVar will be used to generate convenient methods for the collection, such as an adder method addChild(YourType child).
fetch	fetchType	The JPA fetch type for this OneToMany association.

configuration

Table 12.32. configuration properties

Name	Type	Documentation
jdbcConnectivity	jdbcConnectivity	The JDBC settings enabling Celerio to retrieve your database meta data.
databaseInfo	databaseInfo	Specify the database information,

Name	Type	Documentation
		used for documentation only
packs	pack []	List of template packs to execute during the generation. Defaults to the template packs found in the classpath.
modules	module []	List of modules enabled during the generation. Modules are cross cutting functionalities that span across packs.
customModules	string []	List of custom modules enabled during the generation. Modules are cross cutting functionalities that span across packs.
filenames	pattern []	Control the generation output by filtering the generated files based on their filename.
templates	pattern []	Control the generation output by filtering the execution of the generation templates based on their filename.
tables	pattern []	Filter the tables you want to be generated
numberMappings	numberMapping []	The list of number mappings. The first match is used. If no match is found, convention applies.
dateMappings	dateMapping []	The list of date mappings. The first match is used. If no match is found, convention applies.
conventions	conventions	Configure the java convention such as classnames, packages, methods
metaAttributes	metaAttribute []	For future use
generation	generation	Miscellaneous generation configuration
ajax	ajax	Miscellaneous ajax configuration
headerComment	headerComment	The JDBC settings enabling Celerio to retrieve your database meta data.
restriction	restriction	Restrict the generation to the giv-

Name	Type	Documentation
		en elements
associationDirection	associationDirection	Choose the default association direction
applicationName	string	Specify the default application name that is used in the generated pom.xml. It should be one word, no space. Example: casino
rootPackage	string	Specify the default root package for all the generated java code Example: com.mycompany

dateMapping

Global rule to map columns whose JDBC TYPE is DATE, TIME or TIMESTAMP to a Java type.

Table 12.33. dateMapping properties

Name	Type	Documentation
mappedType	mappedType	The mapped type to use when both the jdbcType and the columnNamePattern matches what is expected.
columnJdbcType	jdbcType	Only column with this JdbcType are concerned by this mapping. Accepted JdbcType are DATE, TIME, TIMESTAMP. When set to null, we assume the column JdbcType may be DATE, TIME, or TIMESTAMP.
columnNameRegExp	string	An optional regular expression to restrict the mapping by column name. The matching is case insensitive.

constraintConfig

Defines a constraint configuration. For future usage.

Table 12.34. `constraintConfig` properties

Name	Type	Documentation
`metaAttributes`	metaAttribute []	For future use
`name`	string	Name of the constraint
`logicalname`	string	

`numberMapping`

Global rule to map columns whose JDBC TYPE correspond to a number to a Java type.

Table 12.35. `numberMapping` properties

Name	Type	Documentation
`mappedType`	mappedType	The mapped type to use when both the column size and decimal digit value fall into the specified ranges.
`columnSizeMin`	int	The minimum (inclusive) column size to fall into this mapping range.
`columnSizeMax`	int	The maximum (exclusive) column size to fall into this mapping range.
`columnDecimalDigitsMin`	int	The minimum (inclusive) column decimal digit value to fall into this mapping range.
`columnDecimalDigitsMax`	int	The maximum (exclusive) column decimal digit value to fall into this mapping range.

`classTypeOverride`

Override the class conventions such as GeneratedPackage, suffix and prefixes

Table 12.36. `classTypeOverride` properties

Name	Type	Documentation
`classType`	classType	The ClassType to override
`prefix`	string	Override the prefix for this ClassType
`suffix`	string	Override the suffix for this ClassType
`generatedPackage`	generatedPackage	Override the GeneratedPackage for this ClassType

`ajax`

Table 12.37. `ajax` properties

Name	Type	Documentation
`oneToOne`	boolean	
`manyToOne`	boolean	

`genericGenerator`

Table 12.38. `genericGenerator` properties

Name	Type	Documentation
`parameters`	metaAttribute []	
`name`	string	
`strategy`	string	

`importedKey`

Description of the primary key columns that are referenced by a table's foreign key columns (the primary keys imported by a table).

Table 12.39. `importedKey` properties

Name	Type	Documentation
`fkColumnName`	string	Foreign key column name
`fkName`	string	Foreign key name
`pkColumnName`	string	Primary key column name being imported
`pkTableName`	string	Primary key table name being imported

`manyToManyConfig`

The ManyToManyConfig allows you to fine tune your @ManyToMany association. The ManyToManyConfig element must be a child of a columnConfig element referencing (i.e foreignkey) the entity that is the target of this @ManyToMany association. The columnConfig necessarily belongs to a 'join entity'.

Table 12.40. `manyToManyConfig` properties

Name	Type	Documentation
`cascades`	cascade []	The list of JPA cascade types for the this ManyToMany association.
`var`	string	The variable name for the collection. It should be plural, for example: 'children'.
`elementVar`	string	The variable name for an element of the collection. For example, if the variable name for the collection is 'children', the elementVar should be child. This elementVar will be used to generate convenient methods for the collection, such as an adder method addChild(YourType child).
`fetch`	fetchType	The JPA fetch type for this ManyToMany association.

pattern

A pattern is a structure to help handling inclusion and exclusion of resources

Table 12.41. pattern properties

Name	Type	Documentation
pattern	string	if the pattern contains '?', '*', '**' the matching will be done using an ant matcher, otherwise it will do a equalsIgnoreCase ? matches one character * matches zero or more characters ** matches zero or more 'directories' in a path Some examples: com/t?st.jsp - matches com/test.jsp but also com/tast.jsp or com/txst.jsp com/yourcompany/**\/*.jsp - matches all .jsp files in the com/yourcompany directory
include	boolean	True is is an inclusion pattern, false for an exclusion ?

celerio

Table 12.42. celerio properties

Name	Type	Documentation
includes	include []	For large projects, you can split the content of the entityConfigs tag into multiple files and 'include' the files here.
configuration	configuration	Configure the celerio generator, such as conventions, jdbc connectivity, and other
constraintConfigs	constraintConfig []	Specify constraint configuration (Future use)

Name	Type	Documentation
entityConfigs	entityConfig []	Configure the generated entities.
sharedEnumConfigs	enumConfig []	Configure enums that will be used in multiple entities, and referenced by their name in ColumnConfig

headerComment

Specify your own file header comments

Table 12.43. headerComment properties

Name	Type	Documentation
lines	string []	Set each line to be added to the header files.
include	boolean	Should the header be present in the generated files ?
showTemplateName	boolean	Should the template name be present in the header. This is useful when dealing with large amount of templates and packs for debugging purposes or support information.

generatedValue

Table 12.44. generatedValue properties

Name	Type	Documentation
generator	string	The name of the primary key generator to use
strategy	generationType	The primary key generation strategy that the persistence provider must use to generate the annotated entity primary key.

oneToOneConfig

Table 12.45. oneToOneConfig properties

Name	Type	Documentation
cascades	cascade []	The list of JPA cascade types for the this one-to-one association.
var	string	The variable name for this one-to-one association. It should be singular, for example: 'parent'.
fetch	fetchType	The JPA fetch type for this one-to-one association.
ajax	boolean	Should this many to one be represented as an ajax drop down instead of a simple list ?

methodConventionOverride

change the prefix/suffix conventions for a given method

Table 12.46. methodConventionOverride properties

Name	Type	Documentation
methodConvention	methodConvention	Method type to override Example: GET_LOCALIZED
prefix	string	Override the prefix for this methodConvention Example: get
suffix	string	Override the suffix for this methodConvention Example: Localized

extendsClass

Table 12.47. `extendsClass` properties

Name	Type	Documentation
`fullType`	string	The full class name that this entity extends. For example 'com.mycompany.MyClass'. This is taken into account only if the entity is a root entity.

`columnConfig`

Table 12.48. `columnConfig` properties

Name	Type	Documentation
`usages`	string []	For future uses
`enumConfig`	enumConfig	Specify the enum config to map this column to a Java enum.
`generatedValue`	generatedValue	When the column represents a single primary key, you can configure the GeneratedValue JPA annotation here.
`genericGenerator`	genericGenerator	When the column represents a single primary key, you can configure the GenericGenerator JPA annotation here.
`metaAttributes`	metaAttribute []	for future use
`customAnnotations`	customAnnotation []	List of custom annotations to apply on this property.
`manyToOneConfig`	manyToOneConfig	
`oneToManyConfig`	oneToManyConfig	
`oneToOneConfig`	oneToOneConfig	
`inverseOneToOneConfig`	oneToOneConfig	
`manyToManyConfig`	manyToManyConfig	
`sharedEnumName`	string	References a shared enum name by its name. You cannot have both an enum configuration, and

Name	Type	Documentation
		a shared enum name.
ignore	boolean	If set to true, the column will be ignored. Make sure you do not ignore not null columns.
type	jdbcType	Override the default JdbcType.
mappedType	mappedType	Force the Java mapped type for this column instead of relying on Celerio's conventions.
fieldName	string	The field name, that is the name of the variable. By default, the field name is deduced from the column name. Example: 'first_name' will become 'firstName';
tableName	string	Allows you to use JPA secondary table if you set a table name that is different from the entity table name. Default to the entity table name.
columnName	string	The mandatory column name.
size	int	Override the size defined in the metadata
min	int	Minimum length for String
ordinalPosition	int	Override the ordinal position defined in the metadata
displayOrder	int	The order of appearance of this column in forms, from top to bottom and in search results, from left to right. It defaults to the ordinal position.
typeConverter	string	Specify a type converter for persisting specific columns
businessKey	boolean	Indicates if this property is part of the entity business key. You may set it on several properties at the same time if your business key involves more than one column. If set to true, the property will be used in equals/hashCode methods. As soon as you declare this

Name	Type	Documentation
		attribute on a property, convention no longer applies for the entity.
asTransient	boolean	Allows you to override the getter in a sub-class that extends the base entity. If set to true, all the annotations for the corresponding getter will be commented and a @Transient annotation will be set.
comment	string	The comment that will be inserted as JavaDoc for this column.
decimalDigits	int	Override the decimal digits defined in the metadata
defaultValue	string	Override the default value defined in the metadata
messageKey	boolean	Indicates whether the possible values held by this column are used as keys to resolve the associated localized values.
label	string	The label for this column. It is copied in the entity properties file located in the folder 'src/main/resources/localization/domain-generated'.
inverse	boolean	If this column represents a foreign key that points to the target of a ManyToMany association it can be set to true to change the default inverse side of the ManyToMany association. By convention, the column with the highest ordinal position refers to the inverse side.
associationDirection	associationDirection	If this column represents an importedKey, should it be bidirectionnal or unidirectionnal
enableOneToVirtualOne	boolean	If this column represents an importedKey, and the column is unique, should the one to one be

Name	Type	Documentation
		handled via a collection ?
autoIncrement	boolean	Override the autoIncrement value defined in the metadata. You should use it only in case your driver is unable to determine whether the pk is auto incremented or not.
nullable	boolean	Override the nullable value defined in the metadata
formField	boolean	Should this column be in the form to be filled by your users
searchField	boolean	Should this column be in the search form to be filled by your users
searchResult	boolean	Should this column be present in the search results
selectLabel	boolean	Should this column be part of the label representation
unique	boolean	Override the uniqueness defined in the indexes from the metadata
visible	boolean	Should this column be visible to the users ?
version	boolean	Should this column be used as a version ? This column will be mapped with a @Version
targetTableName	string	Make this column a 'virtual' foreign key, referencing the specified table name. You should not use it if your database schema already declare such constraint.
targetColumnName	string	Once you have set the targetTableName, you can adjust the targetColumnName if it is different from the primaryKey column. Defaults to the targetTableName's primary key column.
targetEntityName	string	If this entity field maps a foreign key column that refers to a table mapped to different entities (i.e. inheritance), you must set the

Name	Type	Documentation
		name of the entity this field refers to.
`targetEntityVar`	string	The variable name used to refer to the target entity.
`sourceEntityVar`	string	DEPRECATED. Please use instead oneToManyConfig child element. The variable name used on the target entity to refer back to this entity. It should be singular.
`m2mVar`	string	DEPRECATED. Please use the manyToManyConfig child element.
`password`	boolean	Should this column be considered as storing a password ? This will impact input types attribute on the web tier.

`cascade`

Table 12.49. `cascade` properties

Name	Type	Documentation
`type`	cascadeType	JPA cascade type.

`table`

Describes all the metadata for a given table

Table 12.50. `table` properties

Name	Type	Documentation
`columns`	column []	Describes all the columns metadata for this table
`indexes`	index []	Describes all the indexes for this

Name	Type	Documentation
		table
importedKeys	importedKey []	Describes all the imported keys for this table
primaryKeys	string []	Describes all the primary keys for this table
name	string	This table name Example: USER
type	tableType	Type of the table
remarks	string	Documentation for this table Example: Table containing all the user related information

generation

Table 12.51. generation properties

Name	Type	Documentation
modelBasePrefix	string	
useMavenCelerioPlugin	boolean	
version	string	
generateCacheAnnotationInEntity	boolean	Tell whether or not the Hibernate @Cache should be generated in Entity. Defaults to true.
caseSensitiveTableAndColumnAnnotations	boolean	Tell whether table/column comparison with entity/property's name is case sensitive. If no, then @Table / @Column annotation may be omitted in certain cases. For example @Table("COUNTRY") would not be generated for @Entity public class Country... as they match. Defaults to false.

xmlFormatter

Table 12.52. `xmlFormatter` properties

Name	Type	Documentation
`enableXmlFormatter`	boolean	Enable Formatter for all XML generated file. Default to false. Note: currently formatting sort attributes in alphabetical order. This is not convenient for certain tags.
`maximumLineWidth`	int	
`indent`	int	

`enumConfig`

Describes an enum class

Table 12.53. `enumConfig` properties

Name	Type	Documentation
`enumValues`	enumValue []	Specify the values that will be added to the current enum
`comments`	string []	Set comments for this enumeration.
`name`	string	Set the name of the generated enum. Example: name="CreditCardEnum"
`rootPackage`	string	Allows you to override the default root package. Example: com.yourcompany
`subPackage`	string	When you define a sub-package, the resulting enum's package becomes "<rootPackage>.domain.<subPackage>" instead of "<rootPackage>.domain". There is no sub-package by default.
`type`	enumType	JPA enum type

Name	Type	Documentation
userType	string	Specifiy the user type implementation to use to be given to hibernate Example: name="com.youcompany.hibernate.support.CustomDateUserType"

jdbcConnectivity

Table 12.54. jdbcConnectivity properties

Name	Type	Documentation
tableTypes	tableType []	Table types to retrieve
driver	string	Jdbc driver name Example: org.h2.Driver
url	string	Jdbc url connection Example: Jdbc:h2:~/mydatabase
user	string	Jdbc user Example: myuser
password	string	Jdbc password Example: mypassword
schemaName	string	
tableNamePattern	string	you can restrict table extraction using a pattern Example: PROJECT_%
oracleRetrieveRemarks	boolean	Should Celerio retrieve remarks on oracle, beware this is a very time consuming operation
oracleRetrieveSynonyms	boolean	Should Celerio retrieve synonyms on oracle
catalog	string	Catalog name; must match the catalog name as it is stored in the database. "" retrieves those without a catalog empty means that the catalog name should not be used to narrow the search

entityConfig

Describes an entity config

Table 12.55. entityConfig properties

Name	Type	Documentation
usages	string []	For future use
metaAttributes	metaAttribute []	For future use
inheritance	inheritance	Inheritance configuration.
extendsClass	extendsClass	Specify the base class that this entity should extends. Only for root entity.
implementsInterfaces	implementsInterface []	Specify the extra interfaces that this entity should implement.
columnConfigs	columnConfig []	This entity's columnConfigs. Note that for entities without inheritance or for entities with a JOIN inheritance strategy, if a column is present in the table's meta data but has no corresponding entityConfig in this list, then an entityConfig is created by default and added automatically to this list.
entityName	string	The JPA entity's type. For example, entityName="BankAccount". By default, the entity name is deduced from the table name. For example: 'bank_account' will become 'BankAccount';
sequenceName	string	Allows you to specify the sequence name to use in order to generate this entity pk value. When a sequence name is provided the corresponding @SequenceGenerator and @GeneratedValue annotations are added to the primary key attribute.

Name	Type	Documentation
`tableName`	string	The underlying table name for the entity. If not set, inheritance must be configured.
`middleTable`	boolean	By convention a table is considered as a many-to-many middle table if it has two foreign keys and no other regular columns. This attribute allows you to consider this table as a middle table, even if it has other regular columns. A regular column is a column that is not used as a primary key or as an optimistic lock.
`comment`	string	The comment that will be inserted in this entity's JavaDoc.
`rootPackage`	string	Allows you to override the default root package. Example: com.yourcompany
`subPackage`	string	When you define a sub-package, the resulting entity's package becomes "<rootPackage>.domain.<subPackage>" instead of "<rootPackage>.domain". There is no sub-package by default.
`associationDirection`	associationDirection	It is pertinent only if this entity's table plays the role of a middle table in a many-to-many association. In that case you can use this parameter to set the many-to-many association direction.
`collectionType`	collectionType	You can override the default collection type for this entity
`label`	string	The label for this entity. It is copied in the entity properties file located in the folder 'src/main/resources/localization/domain-generated'.

column

Configuration of a column, the data reflect the jdbc metadata

Table 12.56. column properties

Name	Type	Documentation
enumValues	string []	Enum values if the column represents an enum
name	string	Column name
columnDef	string	Default value
decimalDigits	int	The number of fractional digits
autoIncrement	boolean	Is Auto Increment?
nullable	boolean	Is NULL allowed ?
ordinalPosition	int	Index of column in table (starting at 1)
remarks	string	Comment describing the column
size	int	Column size. For char or date types this is the maximum number of characters, for numeric or decimal types this is precision.
type	jdbcType	This column jdbc type

metadata

Table 12.57. metadata properties

Name	Type	Documentation
jdbcConnectivity	jdbcConnectivity	
databaseInfo	databaseInfo	
tables	table []	

include

Include a configuration file dedicated to entityConfigs. Use it on large project to split your entityConfigs configuration into smaller pieces.

Table 12.58. `include` properties

Name	Type	Documentation
`filename`	string	The path to a configuration file whose entityConfigs tag will be loaded. The path must be relative to the folder containing the main configuration file. Beware, only the entityConfigs tag will be loaded from this file. For example: includes/ref/country.xml

Examples

Filtering packs

```
<celerio xmlns="http://www.jaxio.com/schema/celerio" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.jaxio.com/schema/celerio ../../../../../../../../../../configuration/target/jibx/celerio.xsd">
  <configuration>
    <packs>
      <pack name="pack-backend" enable="true" />
      <pack name="pack-mvc-common" enable="false" />
      <pack name="pack-jsf" enable="false" />
    </packs>
  </configuration>
</celerio>
```

Filtering filenames

```
<celerio xmlns="http://www.jaxio.com/schema/celerio" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.jaxio.com/schema/celerio ../../../../../../../../../../configuration/target/jibx/celerio.xsd">
  <configuration>
    <filenames>
      <filename include="true" pattern="**/sql/**/*.sql" />
    </filenames>
  </configuration>
</celerio>
```

Filtering tables

```
<celerio xmlns="http://www.jaxio.com/schema/celerio" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.jaxio.com/schema/celerio ../../../../../../../../../../configuration/target/jibx/celerio.xsd">
  <configuration>
    <tables>
      <table include="false" pattern="role" />
    </tables>
  </configuration>
</celerio>
```

Filtering templates

```
<celerio xmlns="http://www.jaxio.com/schema/celerio" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.jaxio.com/schema/celerio ../../../../../../../../../../configuration/target/jibx/celerio.xsd">
      <configuration>
            <templates>
                  <template include="true" pattern="**/resources/log4j.p.vm.*" />
                  <template include="true" pattern="src/main/resources/spring/**" />
            </templates>
      </configuration>
</celerio>
```