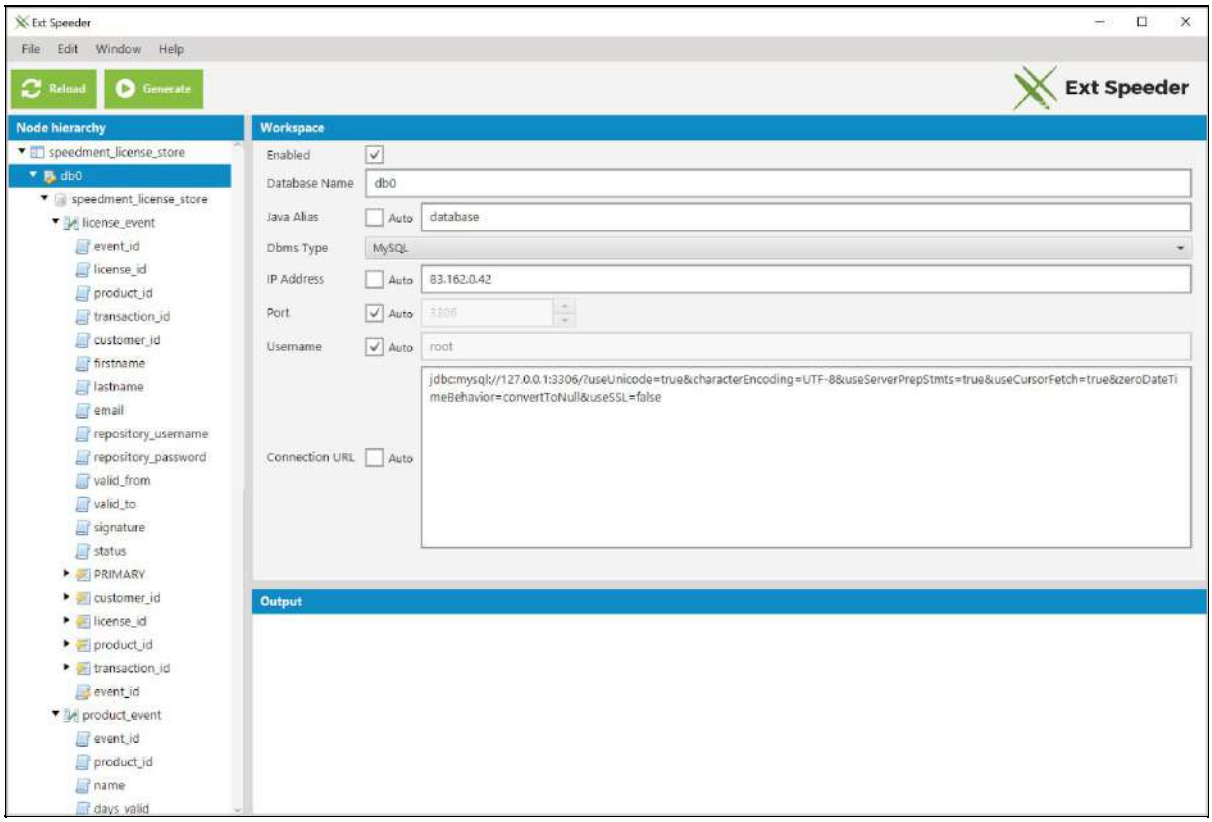
Ext Speeder

User's Guide

Ext Speeder version 1.0.6



Speedment

Speedment, Inc.
Palo Alto, California, United States

Table of Contents

1 Introduction

With the *Ext Speeder* add-on for Sencha Ext JS, it is possible to automatically generate an API bridge application between a Sencha Ext JS Grid application and a relational database.

1.1 Scope and Purpose

This User's Guide will take you through the process of setting up Ext Speeder environment and configuring it for maximum performance. It will also give an introduction to the Ext Speeder graphical tool and to the generated REST API.

Ext Speeder runs on the server-side of the application architecture and responds to Ext JS DataGrid requests over a REST API. The examples are aimed towards Sencha Ext JS developers that works with database communication. The back-end code is written in Java, but in most use cases, the amount of code that needs to be written manually is minimal. Ext Speeder will take care of all the boilerplate code.

1.2 Prerequisites

The following software needs to be installed on a development computer:

- Java Development Kit (JDK) 1.8.0_40 or newer (Oracle or OpenJDK)
- One of the following IDEs (with Maven support):
  - Netbeans IDE 8.1
  - Eclipse IDE Mars.1
  - IntelliJ IDEA 16.1
- Maven 3.3.0 or later
- Sencha Ext JS 6.0.0 or later

An internet connection is also required the first time an Ext Speeder license is setup.

1.3 Quick Start

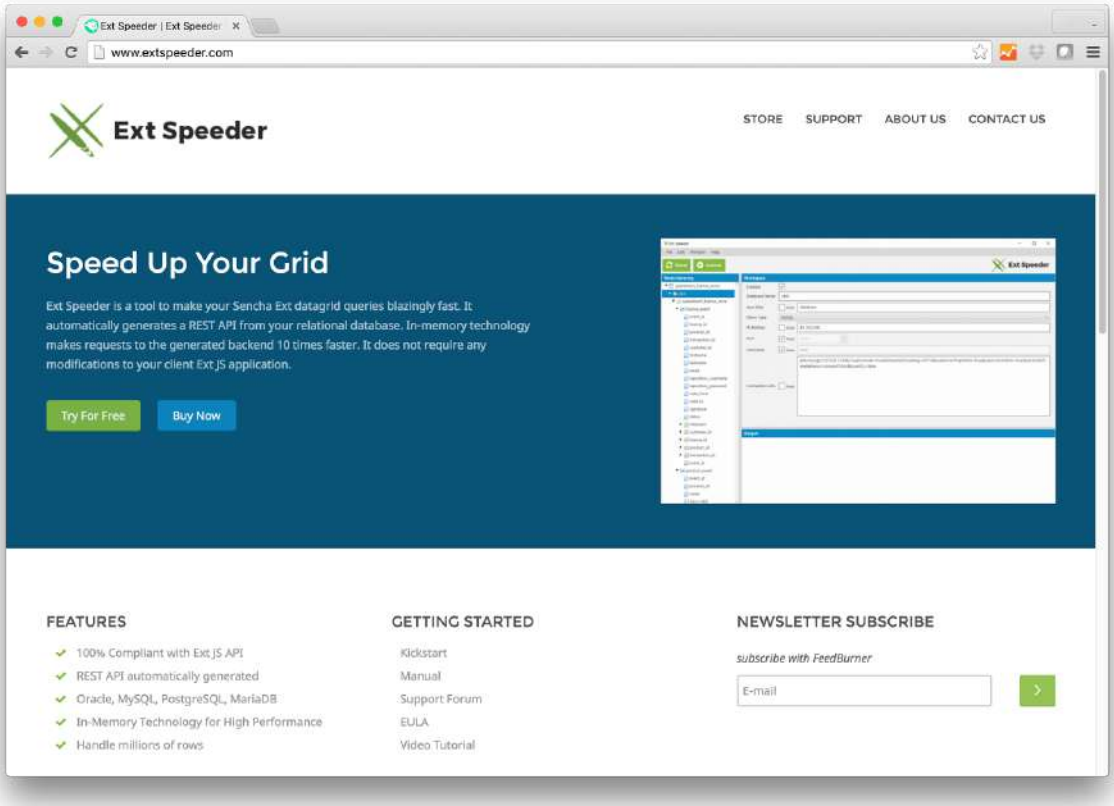
In Chapter 1.4 to 1.7 you will get a detailed description of how to install and use Ext Speeder. Here is an overview of the work-items that needs to be completed:

1. Modify the **~/.m2/settings.xml** file so that the Ext Speeder repositories will be recognized.
2. Launch your IDE (Netbeans, Eclipse or IntelliJ IDEA).
3. Create a new project.
4. Modify the new project's POM file to allow it to use Ext Speeder.
5. Launch the Ext Speeder tool and follow the instructions given: select database and click **Connect**.
6. Click the **Generate** button. Code will be generated automatically.
7. Right-click the **Generated{$ProjectName}Application** file and select "**Run File**".

Step 1 needs to be completed only once. Step 2 through 7 are repeated for each new project that is being created. The steps are different depending on the IDE type used.

1.4 Getting the Software

The first step in getting access to the Ext Speeder tool is to register for a license. Go to www.extspeeder.com and click on **Buy Now** or **Try For Free** to get to the online store. Once the form is completed, a product key, a username and a password will be delivered to the specified e-mail address.



The first time Ext Speeder is launched from within the IDE, the product key needs to be entered in order to start the tool.

Ext Speeder runs inside any of the IDEs: NetBeans, Eclipse or IntelliJ IDEA. These tools will automatically download the Ext Speedment tool using the included Maven builder. However, Maven first has to be configured to find the Ext Speeder repositories. To do this, a file called “settings.xml” needs to be updated.

Navigate to your home directory and look for a folder named “.m2/”. The folder might be hidden by default, so make sure to show hidden files and folders. Inside “.m2/”, open the file “settings.xml” in a text editor.

Tip: If Maven has not been used before, there is a probably not a “.m2” directory and/or a “settings.xml”-file. In that case, create the missing directory manually and then use a text-editor to create a new empty “settings.xml” file.

Update the “settings.xml”-file so it will look like the file below, but replace “*USERNAME*” and “*PASSWORD*” with the information sent to you in the welcome e-mail.

```
<settings>
    <servers>

        <server>
            <id>speedment-enterprise</id>
```

```
            <username>USERNAME</username>
            <password>PASSWORD</password>
        </server>
        <server>
            <id>speedment-enterprise-snapshots</id>
            <username>USERNAME</username>
            <password>PASSWORD</password>
        </server>

    </servers>

    <profiles>
      <profile>
        <activation>
          <activeByDefault>true</activeByDefault>
        </activation>
        <repositories>
          <repository>
            <id>speedment-enterprise-snapshots</id>
            <name>Speedment Enterprise Snapshot Repositories</name>
    <url>http://repo.speedment.com/nexus/content/repositories/snapshots/</url>
          </repository>
          <repository>
            <id>speedment-enterprise</id>
            <name>Speedment Enterprise Repositories</name>
    <url>http://repo.speedment.com/nexus/content/repositories/releases/</url>
          </repository>
        </repositories>
        <pluginRepositories>
          <pluginRepository>
            <id>speedment-enterprise-snapshots</id>
            <name>Speedment Enterprise Snapshot Repositories</name>
    <url>http://repo.speedment.com/nexus/content/repositories/snapshots/</url>
          </pluginRepository>
          <pluginRepository>
            <id>speedment-enterprise</id>
            <name>Speedment Enterprise Repositories</name>
    <url>http://repo.speedment.com/nexus/content/repositories/releases/</url>
          </pluginRepository>
        </pluginRepositories>
      </profile>
    </profiles>
</settings>
```

This will tell Maven where the Ext Speeder repositories are located and which credentials are to be used when connecting to the Ext Speeder repositories.

1.5 Creating a New Project

Below are detailed instructions for how to set up a project in each of the supported IDE:s. Skip to the relevant chapter depending on the IDE prefered.

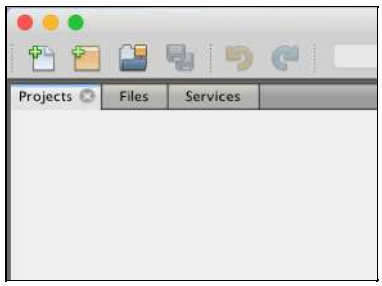
1.5.1 NetBeans IDE

So far the Ext Speeder repositories have been installed together with the credentials in the “settings.xml”-file. It is now time to create a new project.

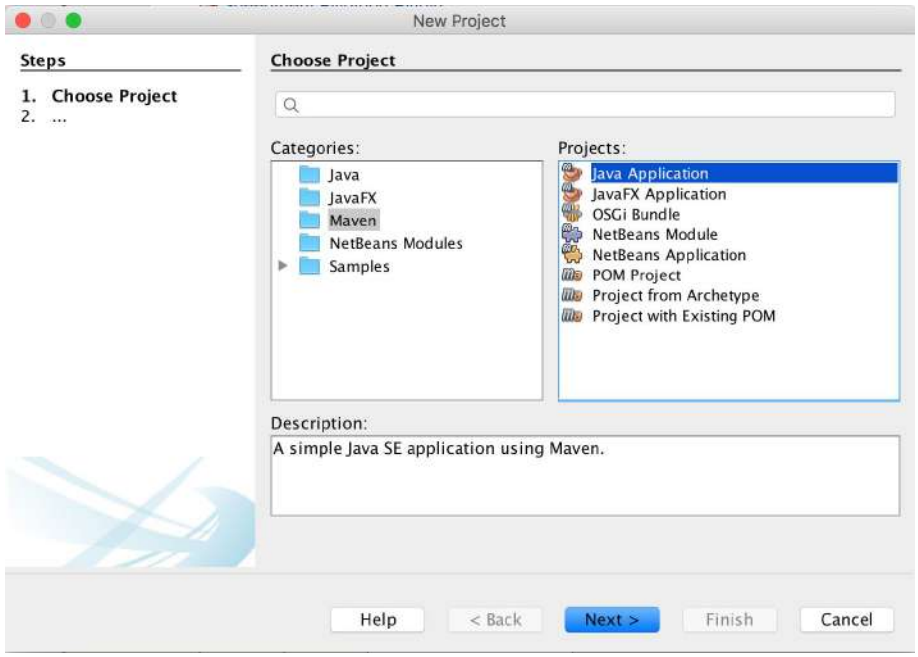
1.5.1.1 Creating a new Ext Speeder Project in Netbeans

At this point the first Ext Speeder project can be created.

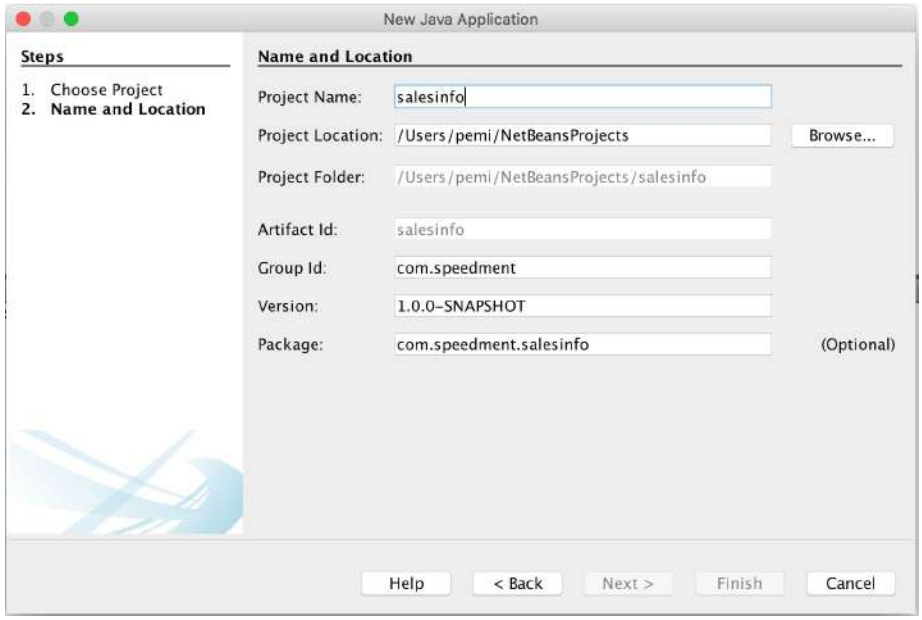
1. The first step of creating a new Ext Speeder project in Netbeans is to press the **New Project...** button in the toolbar.



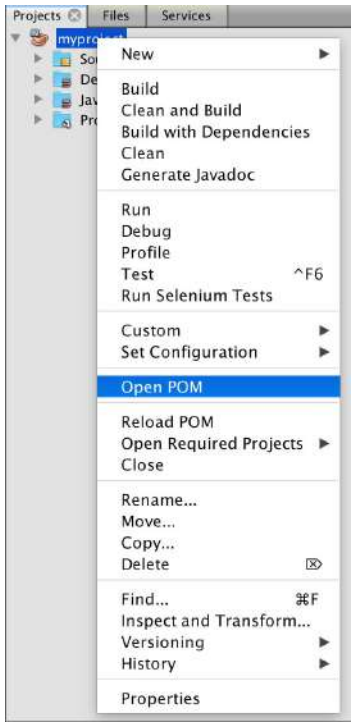
2. This will open a guide for creating a new project. Ext Speeder is made for the Maven build tool, so go to the **Maven** category and select **Java Application**. Press **Next**.



3. Select a **Project Name** and perhaps set other properties for the new server application. Any values can be used. Press **Finish**.



4. The new project is created. Right click on the new project and select **Open POM**.



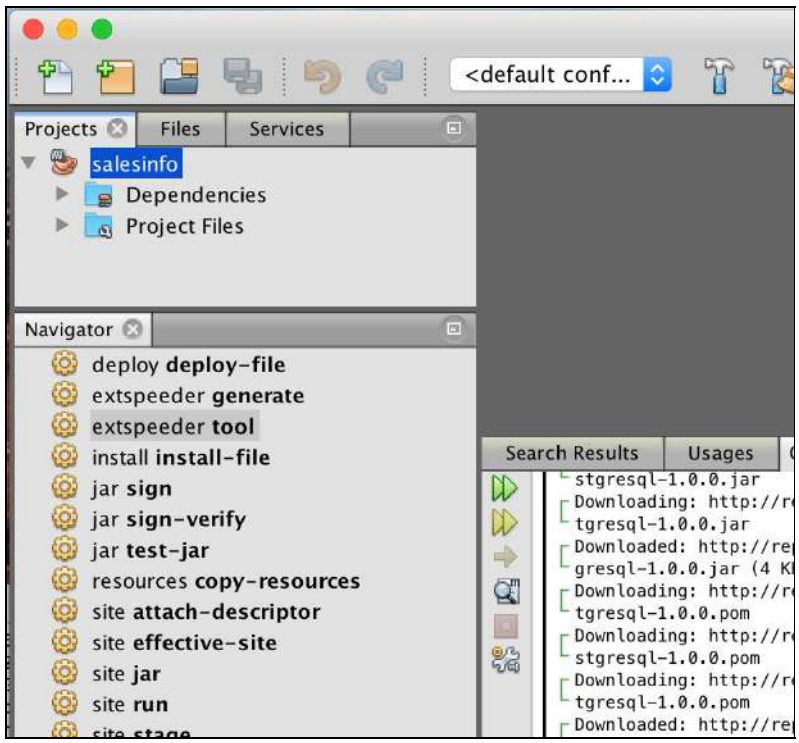
5. Update the new project's POM file so that it contains the Ext Speeder dependencies. You can find these at www.extspeeder.com/userguide.

1.5.1.3 Starting the Ext Speeder Tool from Netbeans

Ext Speeder is designed to run as a Maven Goal inside the IDE. To launch it, follow the steps below:

1. Select the new project in the **Projects** tree. Right-click on the project and select **Clean and Build...** Netbeans will now download all the Ext Speeder components.

2. Look in the **Navigator** panel. Two goals named “extspeeder generate” and “extspeeder tool” will appear.



   Tip: If the Navigator component is not visible, it can opened from the menu bar by pressing **Window → Navigator** or by pressing **Ctrl+7** on the keyboard. If it is showing but does not contain anything, it means that the project root node is not selected in the **Projects** component.

3. Double click on **extspeeder tool** to launch the graphical tool.

   Tip: It is possible to regenerate the project without launching the graphical tool by running the **extspeeder:generate** goal. It will rerun the most recent configuration.

Go to Chapter 1.6 for instructions on how to use the graphical tool.

1.5.2 Eclipse IDE

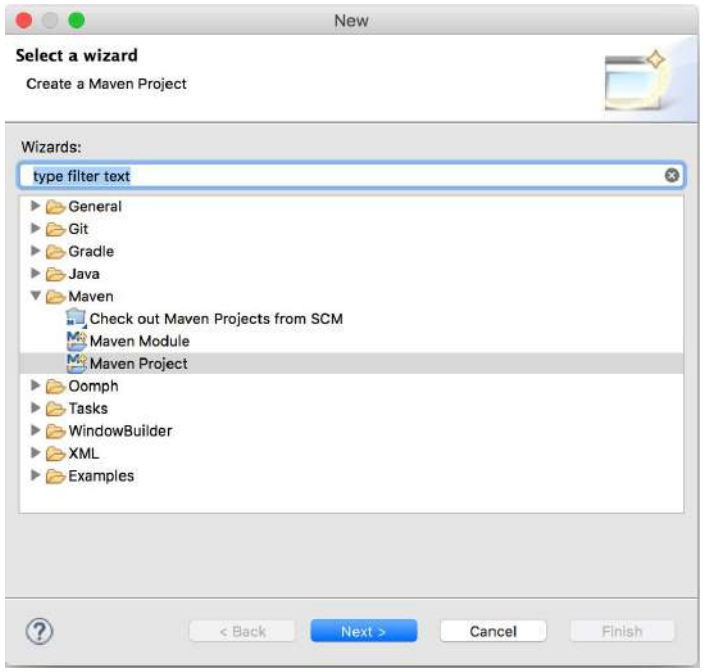
So far, the Ext Speeder repositories and credentials has been installed in the "settings.xml"-file. It is now time to create a new project.

1.5.2.1 Creating a new Ext Speeder Project in Eclipse

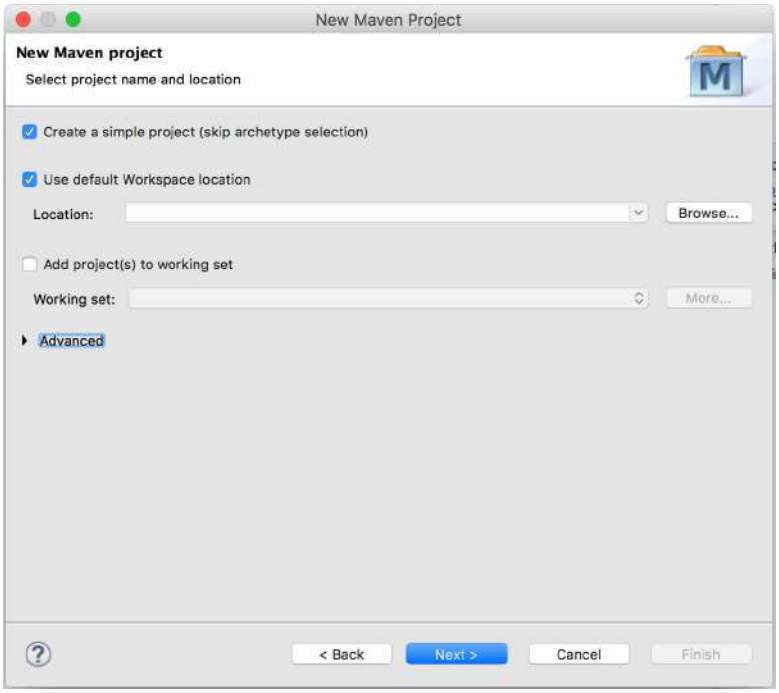
1. Open up a new Eclipse Workspace where the Ext Speeder project is to be located.

2. Go to the **Create Project** button in the toolbar, press the small arrow-down icon next to it and select **Other...**

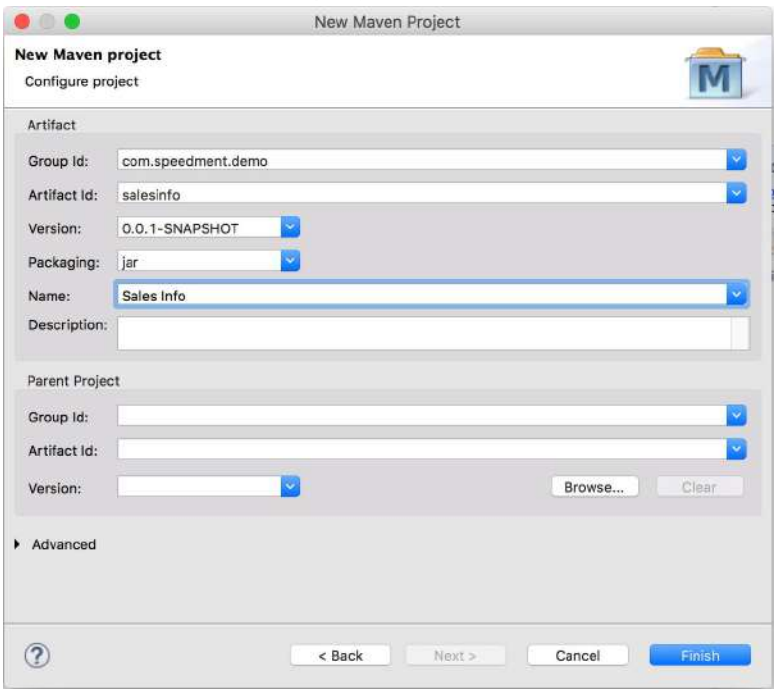
3. Select the **Maven Project** Wizard. Click **Next**.



4. Make sure that the checkbox “Create a simple project...” is checked. Click **Next**.



5. Fill in the data for your project . Any values can be used. Click **Finish**.



6. The project is created. Expand the project and double click on the file “pom.xml”. Select the tab “pom.xml” in the lower portion of the editor. Update the new project’s POM file so that it contains the Ext Speeder dependencies. You can find these at www.extspeeder.com/userguide.

1.5.2.2 Starting the Ext Speeder Tool from Eclipse

Ext Speeder is designed to run as a Maven Goal inside the IDE, but Eclipse does not have a list of executable goals like other IDEs have. Instead, an action for running the tool must be created manually.

1. The first time the Ext Speeder Tool is used, press the small arrow to the right of the **Run** icon in the Eclipse toolbar and select **Run As → 3 Maven Build** or press **Alt+Shift+X, M** on the keyboard.

2. Enter a **Name:** for the goal that can be recognized to be the Ext Speeder Tool, like "ext-speeder-tool".

3. Enter a **Base directory:** for the goal to run in. The easiest way of doing this is to press the **Browse Workspace...** button and select the root node there.

4. Enter "extspeeder:tool" as the **Goals:** to run.

5. Click **Apply** to save the action.To run the tool immedietly, press **Run**.

6. To run the “extspeeder:tool”-goal again, simply press the arrow-down icon beside the **Run** button in the toolbar and select the new action. This will relaunch the Ext Speeder tool.

Go to Chapter 1.6 for instructions on how to use the graphical tool.

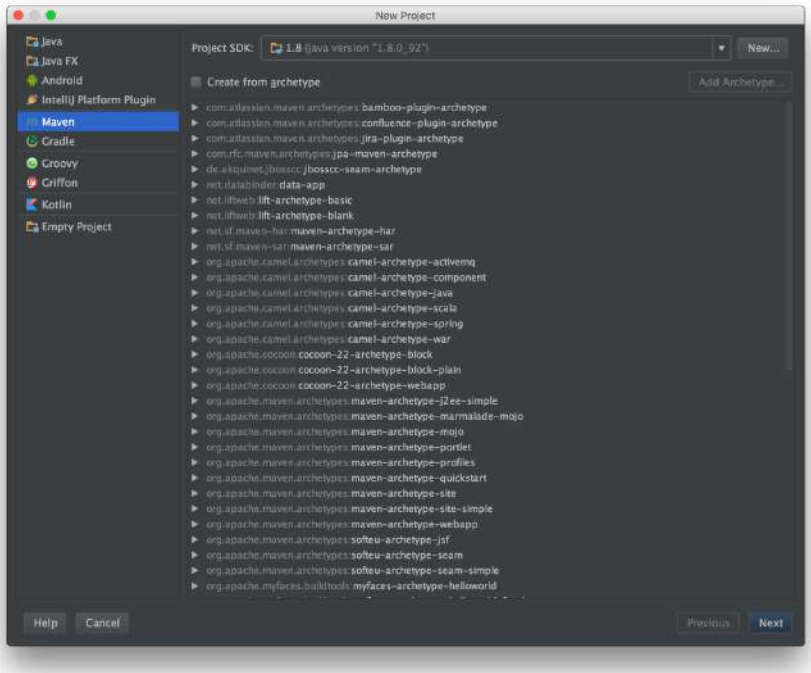
1.5.3 IntelliJ IDEA

So far the Ext Speeder repositories and credentials has been installed in the “settings.xml”-file. It is now time to create a new project.

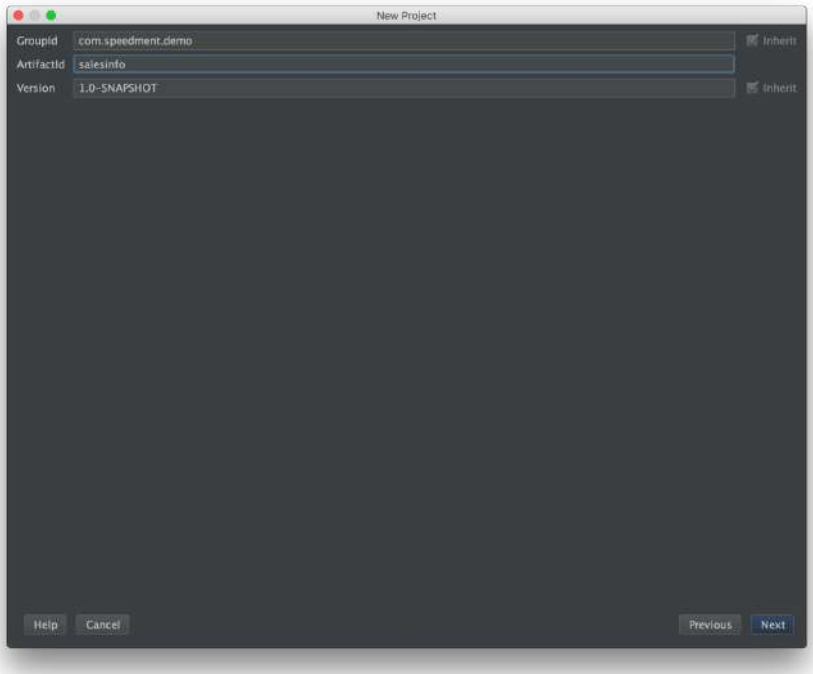
1.5.3.1 Creating a new Ext Speeder Project in IntelliJ IDEA

1. Go to **File → New → Project...**

2. Make sure you are under the **Maven** tab, make sure that the **Project SDK** is 1.8 and that the **Create from archetype** checkbox is unchecked. Click the **Next** button.

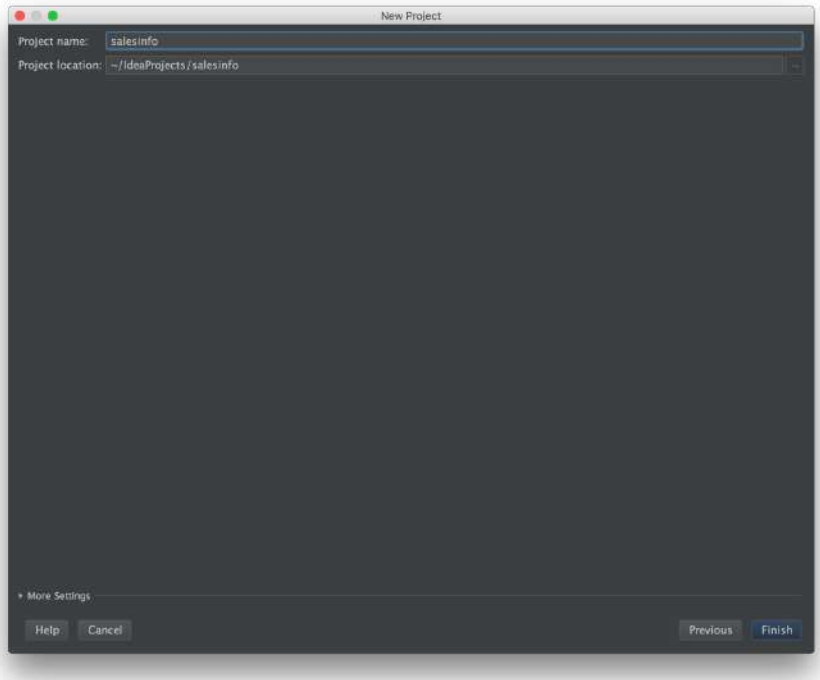


3. Enter the **Goupid**, **ArtifactId** and **Version** of the new project. Any values can be used.

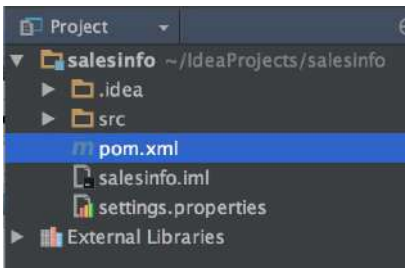


When you have entered the properties, click **Next**.

4. Enter the **Project name** and the **Project location**. Any values can be used. Click **Finish**.



5. You will now return to the workspace of IntelliJ IDEA. Expand the newly created project by clicking the arrow on the left side of the project name.



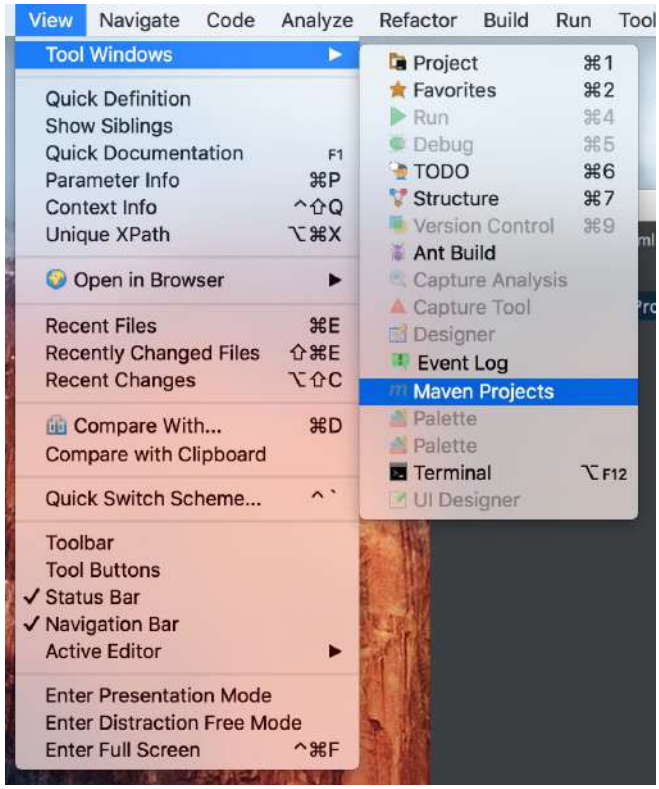
6. Double-click the file named "**pom.xml**". Update the new project's POM file so that it contains the Ext Speeder dependencies. You can find these at www.extspeeder.com/userguide.

7. In the top-right corner of the window you might notice a message with the text "Maven projects need to be imported". Click on **Import Changes** it to import the Ext Speeder dependencies. You project is now ready for Ext Speeder!

1.5.3.2 Starting the Ext Speeder Tool from IntelliJ IDEA

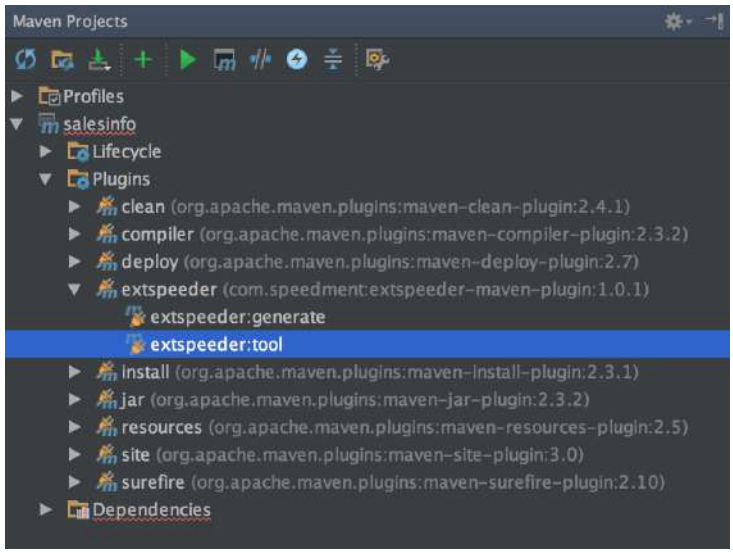
Ext Speeder is designed to run as a Maven Goal inside the IDE. To launch it, follow the steps below:

1. In the far right of the IntelliJ window you should see a tab labeled **Maven Projects**. Maximize it to see the tree of Maven Goals. If the tab is not visible, turn it on by selecting the menu **View->Tool Windws->Maven Projects.**



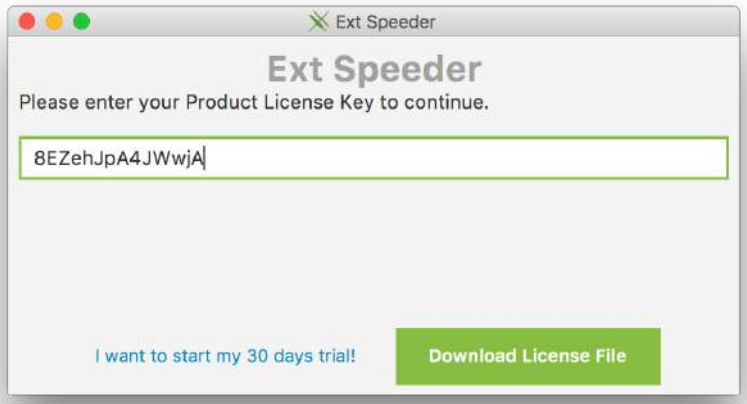
2. Navigate to **<project-name>** → **Plugins** → **extspeeder**.

3. Double-click the **extspeeder:tool** goal.



1.6 Connect to the Database

1. When the Ext Speeder graphical tool is launched for the first time, it will ask for the product key. This is the code that was sent via e-mail when the license was registered with the Ext Speeder website.

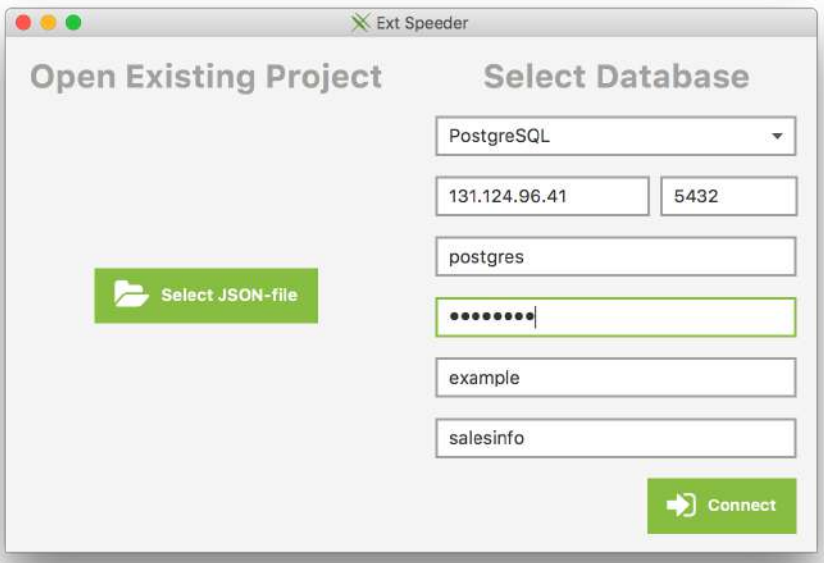


Note: Replace the license key illustrated above with the real one received in the e-mail.

2. Enter e-mail address, firstname and lastname in order for the license to be issued for this computer. This information will be sent to the Ext Speeder servers so that a personal license token can be generated for you (make sure that the development computer is connected to the internet at this time). The token will then be stored on your computer so that you can access the features of Ext Speeder, even without an internet connection in the future.

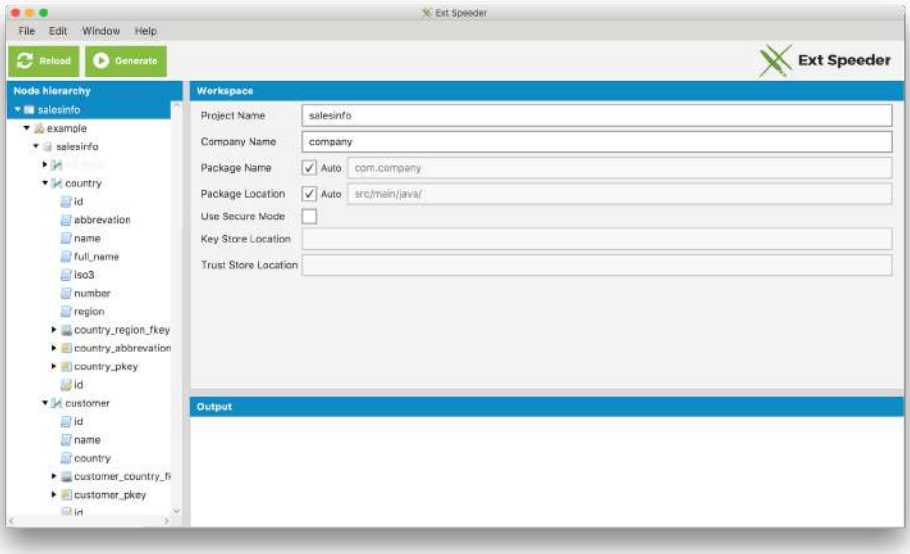
Tip: The same license can be used on multiple computers by entering the same information again. If additional licenses are needed or if a license needs to be transferred to another developer, please contact the Ext Speeder Sales team.

3. The next step is to connect to the database. In the drop-down list at the top, all the currently installed database connectors will be seen. Select the one that is relevant for the project and then continue by entering the credentials for that database in the remaining fields. Add the name of the database schema that should be used in the field at the very bottom of the popup window.

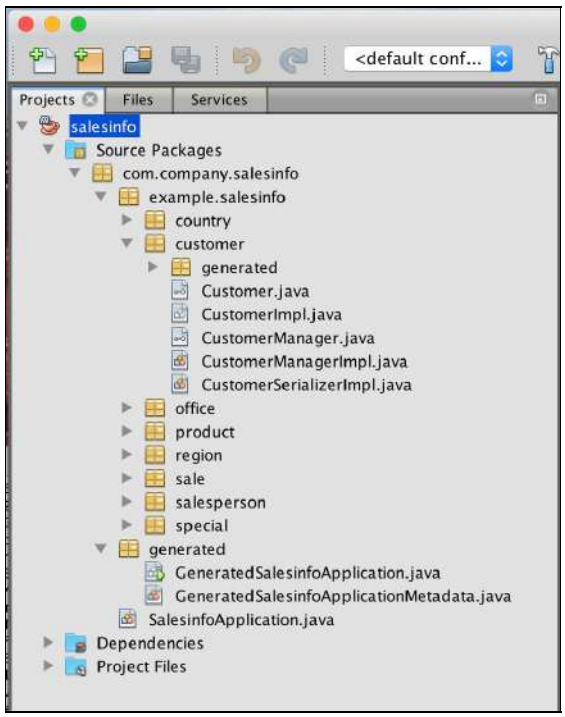


Note: Ext Speeder never saves a database password for security reasons.

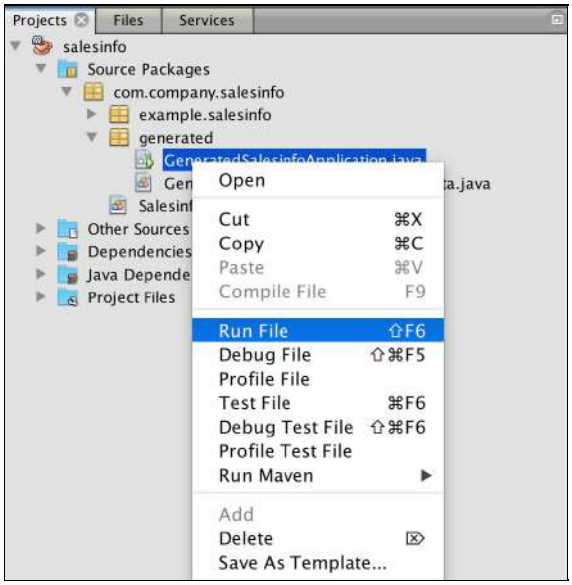
4. After entering the information elements, the Ext Speeder graphical tool will open and the schema will load. Press the **Generate** button in the toolbar to generate code.



5. In the IDE, a number of .java-files and packages that have been generated can be seen. Ext Speeder has created an object-oriented model of the database. In the base package there is now a file named after the project. It includes all the code necessary to start a server that will connect to the database and serve any requests to it via a REST API.



6. Build and run the java project to launch the server application by right-clicking the Generated{$ProjectName}Application file and select “Run”.



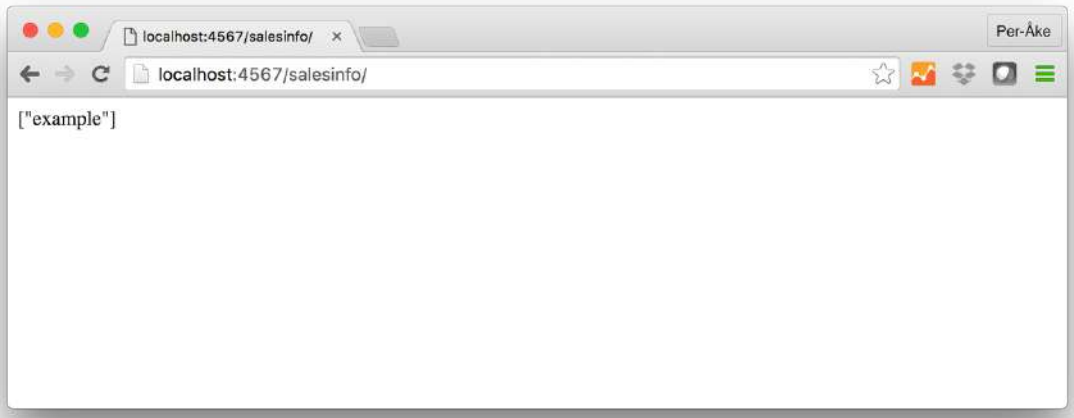
Note: When the application starts, enter a username and a password to connect to the database. If the database is not password protected, simply leave the password field blank.

```
------------------------------------------------------------------------
Building salesinfo 1.0.0-SNAPSHOT
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ salesinfo ---
Warning! You are running from an integrated console.
Entered passwords will be visible in input.
Connecting to database
2016-05-02T22:57:24.598Z INFO  [main] (c.s.e.c.u.Announcer) - Enterprise Component
OffHeapReadOnlyCacheComponentImpl created.
Connect to PostgreSQL on 130.211.120.81:5432
Username (leave blank to use "postgres"):
Password (leave blank to skip password): C4o4F428
Starting server at /salesinfo ...
2016-05-02T22:58:08.164Z INFO  [main] (c.s.i.c.r.SpeedmentApplicationLifecycle) - PostgreSQL, 9.4.5,
PostgreSQL Native Driver PostgreSQL 9.4 JDBC4 (build 1205), JDBC version 4.0
2016-05-02T22:58:08.344Z INFO  [ForkJoinPool.commonPool-worker-5]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed special, 0 entities in 106.84 ms
2016-05-02T22:58:08.410Z INFO  [ForkJoinPool.commonPool-worker-5]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed product, 16 entities in 59.45 ms
2016-05-02T22:58:08.410Z INFO  [ForkJoinPool.commonPool-worker-5] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for special
2016-05-02T22:58:08.412Z INFO  [ForkJoinPool.commonPool-worker-5] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for product
2016-05-02T22:58:08.554Z INFO  [ForkJoinPool.commonPool-worker-1]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed salesperson, 20 entities in 318.42
ms
2016-05-02T22:58:08.554Z INFO  [ForkJoinPool.commonPool-worker-1] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for salesperson
2016-05-02T22:58:08.556Z INFO  [ForkJoinPool.commonPool-worker-6]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed region, 8 entities in 307.55 ms
2016-05-02T22:58:08.556Z INFO  [ForkJoinPool.commonPool-worker-6] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for region
2016-05-02T22:58:08.560Z INFO  [ForkJoinPool.commonPool-worker-4]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed customer, 142 entities in 316.86 ms
2016-05-02T22:58:08.560Z INFO  [ForkJoinPool.commonPool-worker-4] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for customer
2016-05-02T22:58:08.561Z INFO  [ForkJoinPool.commonPool-worker-2]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed sale, 42 entities in 303.29 ms
2016-05-02T22:58:08.561Z INFO  [ForkJoinPool.commonPool-worker-6] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for sale
2016-05-02T22:58:08.566Z INFO  [ForkJoinPool.commonPool-worker-3]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed office, 61 entities in 309.98 ms
2016-05-02T22:58:08.567Z INFO  [ForkJoinPool.commonPool-worker-3] (c.s.e.o.i.c.EntityCache) - Building
```

```
Column Caches for office
2016-05-02T22:58:08.587Z INFO  [ForkJoinPool.commonPool-worker-7]
(c.s.e.o.i.c.i.OffHeapReadOnlyCacheComponentImpl) - Loading completed country, 246 entities in 326.85 ms
2016-05-02T22:58:08.587Z INFO  [ForkJoinPool.commonPool-worker-7] (c.s.e.o.i.c.EntityCache) - Building
Column Caches for country
2016-05-02T22:58:08.627Z INFO  [main] (c.s.i.c.r.SpeedmentApplicationLifecycle) - Java Platform API
Specification 1.8 by Oracle Corporation. Implementation Oracle Corporation 1.8.0_92 by Oracle Corporation
2016-05-02T22:58:08.627Z INFO  [main] (c.s.i.c.r.SpeedmentApplicationLifecycle) - Speedment (Open Source)
version 2.3.2 by Speedment Inc. started. API version is 2.3
Waiting for connections on port 4567...
Press Enter to close application.[Thread-4] INFO org.eclipse.jetty.util.log - Logging initialized
@44938ms
[Thread-4] INFO spark.webserver.JettySparkServer - == Spark has ignited ...
[Thread-4] INFO spark.webserver.JettySparkServer - >> Listening on 0.0.0.0:4567
[Thread-4] INFO org.eclipse.jetty.server.Server - jetty-9.3.2.v20150730
[Thread-4] INFO org.eclipse.jetty.server.ServerConnector - Started
ServerConnector@73faeaac{HTTP/1.1,[http/1.1]}{0.0.0.0:4567}
[Thread-4] INFO org.eclipse.jetty.server.Server - Started @45008ms
```

7. To test the API, launch any web browser and enter the address `http://localhost:4567/<project-name>/` and a list of available command(s) will be shown.

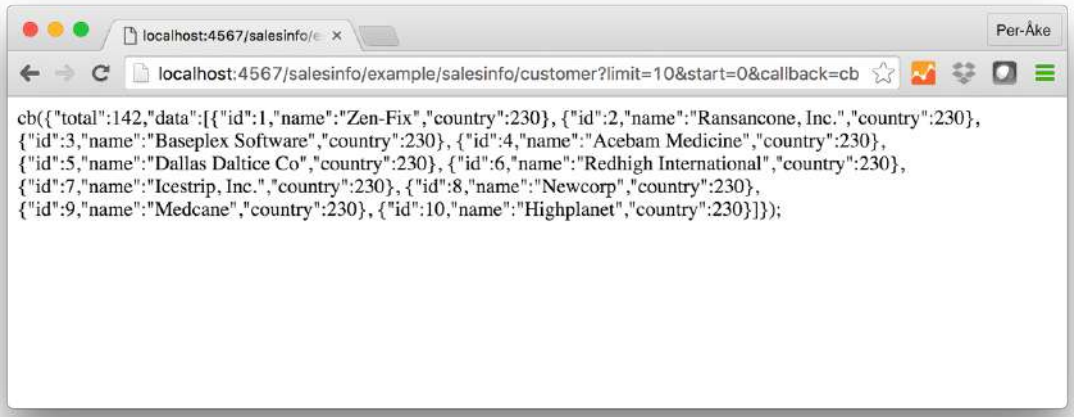


8. These commands can be appended to the URL to request more specific information from the API.

   For example, entering...

   ```
   http://localhost:4567/<project-name>/<database-name>/
   <schema-name>/<table-name>?limit=10&start=0&callback=cb
   ```

   ...will retrieve the first ten rows of data from the given table.



1.7 Configure the Sencha Ext JS Client Project

Ext Speeder is now installed on the development computer and the server project has been created. The default settings in the graphical tool will simply expose every table in the selected schema as separate REST URI path.

Open the Sencha Ext JS client application folder and navigate to the store that the Data Grid is using. A typical model and store pair could look like this:

```
Ext.define('MyApp.model.User', {
  extend: 'Ext.data.Model',
  fields: ['name', 'email', 'phone']
});

Ext.define('MyApp.stores.Users', {
  extend: 'Ext.data.Store',
  model: 'MyApp.model.User',
  data: [
    {id: 1, name: 'Ann', email: 'ann@company.com'},
    {id: 2, name: 'Bill', email: 'bill@company.com'},
    {id: 3, name: 'Cecil', email: 'cecil@company.com'},
    {id: 4, name: 'Dave', email: 'dave@company.com'}
  ]
});
```

The store must now be modified so that it will, instead of using static data, connect to the server application using the generated REST API.

```
Ext.define('MyApp.store.Users', {
    extend: 'Ext.data.BufferedStore', // Buffer requests in client
    model : 'MyApp.model.User',
    proxy : {
        // load using HTTP
```

```
        type        : 'jsonp',
        url         : 'http://localhost:4567/myapp/db0/myapp/user',
        callbackKey : 'callback',
        reader: {
            rootProperty:  'data',  // Default in Ext Speeder
            idProperty:    'id',    // Primary Key Column of table User
            totalProperty: 'total'  // Default in Ext Speeder
        }
    },
    autoLoad : true,
    pageSize : 50,
    sorters  : [{
        property  : 'id',
        direction : 'ASC'
    }]
});
```

Note that the default URL will depend on the names used for the project, database, schema and table. Also, make sure that the 'idProperty' value ('id' in the example above) corresponds to the name of the primary key column for the specific table.

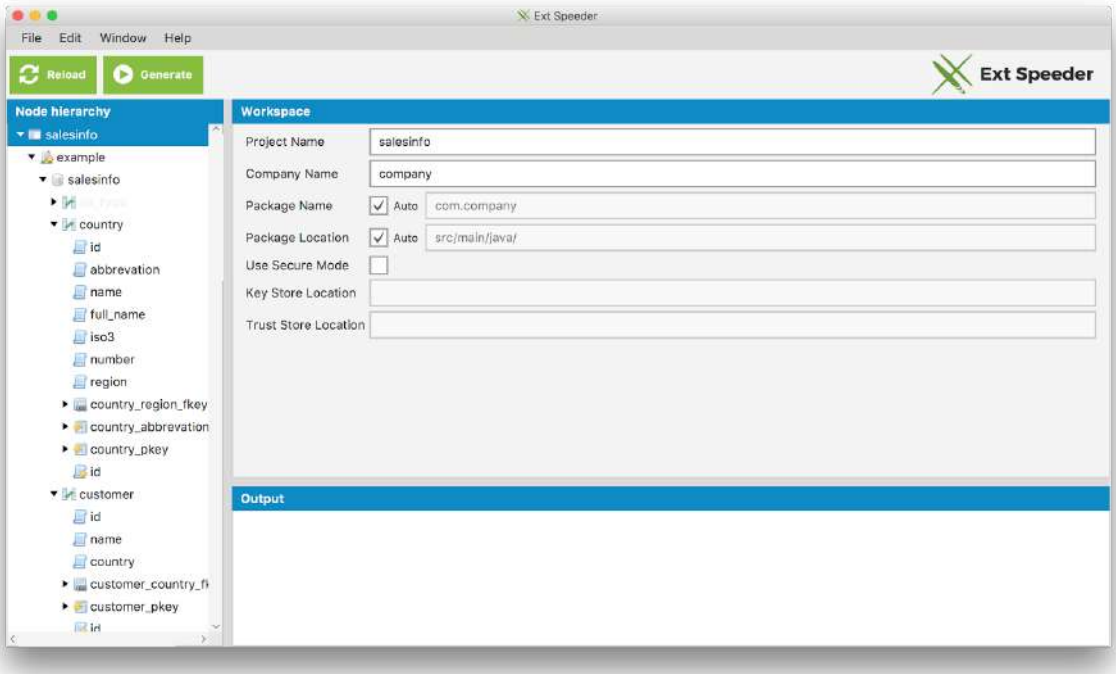
Tip: It is possible to set a custom path for a particular resource using the Ext Speeder graphical tool.

2 Using the Graphical Tool

In the previous chapter a quick look at the graphical tool when creating the project was shown. When the code was generated, Ext Speeder used the default settings for configuring the entire API. This is great to get an API up and running with minimal effort, but sometimes it is necessary to do small adjustments of, for an example, the naming of the commands. This is what we are going to do now.

Using the Java IDE, run the **extspeeder:tool** goal again to open up the main window.

Note: The first time code is generated using Ext Speeder, the configuration will be stored in a .json-file (named “speedment.json”) in the project directory. When the tool is subsequently launched, it will not connect to the database, but instead read the configuration from that .json-file. To regenerate the project from the database, use the **Reload**-button in the **Toolbar** or remove the “speedment.json” file altogether and re-start the tool.

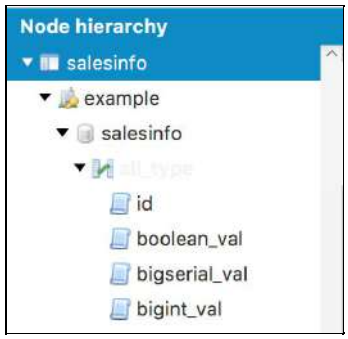


The main window is separated into four different areas. There is the **Toolbar**, the **Project Tree**, the **Workspace** and the **Output** area.

1. The **Toolbar** is located in the upper part of the window. Here it is possible to access all the built-in functions. It is also possible to manage the project, generate code, toggle visibility of different areas and get information about the tool. The

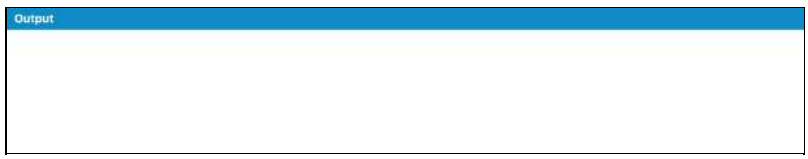
most common tasks can be done using the larger quick-access buttons.

2. The **Project Tree** illustrates a hierarchical representation of the database. By selecting different nodes in the tree, the context of the workspace can be changed. It is also possible to right-click on the nodes to show specific options for that node.



3. The **Workspace** is where most of the configuration is done. The context is based on which node is selected in the Project Tree. Different nodes can have different configuration options. More about that in section 4.1.

4. The **Output** area is where information collected by the generator is displayed.



2.1 Configuring the Project

When the database is analyzed, Ext Speeder will set every option to a default value. This works in most cases, but sometimes a few things have to be changed to match the client API or to hide some piece of information from being exposed. Since every node can contain different configuration options, they are listed below based on which node they belong to.

Tip: If the mouse pointer is hovered over a configuration option in the Workspace, a small description of that setting will appear.

Tip: There are typically two names for every node used by Ext Speeder; a *database name* and a *java alias*. For most purposes, only the **java alias** should be changed and not the database name. The only time a new name of the database is needed is when adapting to structural changes in the database (e.g. a column has been renamed).

2.1.1 Common Node Properties

Most of the node types have a handful of common fields that has the same meaning for all of the node types. These includes:

Property	Meaning
Enabled	Controls if this node shall be handled by the Ext Speeder or not. Disabled nodes will neither be seen in the REST interface nor occupy memory resources in the application. Make sure to disable nodes that are not used by the application. This function is also useful when 'hiding' columns (e.g. password or credit card numbers) from exposure.
Database Name	Represents the name that this node has in the database. For example the name of a database table or a column. Do not change the Database Name unless the name in the database itself has changed.
Java Alias	If the Java Alias checkbox 'auto' is selected, then the Java Alias will be the same as the Database Name. If the 'auto' box is unchecked, then a custom name can be set. The application will use the custom name in the Java code and in the REST API. This is useful if the database structure cannot be changed.

2.1.1.1 Project

The Project Node is the root of the project in the graphical tool. It contains general properties of the project. Properties set here relates, for example, to the Java code generator so it knows where to store the .java-files and which package names to use.

Property	Meaning
Company Name	Holds the name of your company or organization. The Company name is used to automatically derive the Package Name.
Package Name	If the checkbox 'auto' is enabled, the Package Name will be automatically derived as 'com.{Company Name}'. If the the 'auto' box is unchecked, any value can be entered. The

	Package Name is used when organizing the Java files in a name space. The Package Name can, for example, be set to “org.somecharity” or “edu.someuniversity”.
Package Location	If the checkbox ‘auto’ is enabled, the Package Location will be automatically derived as ‘src/main/java’ which is Maven’s recommended location for Java files. If the ‘auto’ box in unchecked, any value can be entered (only for advanced users).
Cache Storage Type	This drop-down list controls how Ext Speeder accesses data. Selecting the right type is essential for application performance.
● NO_CACHE	Data will be pulled directly from the database with no cache in between the database and the application. Data will always be current, but no speed increase is gained.
● STATIC_ON_HEAP	Data is pulled from an in-memory on heap store. This is the fastest solution, but the amount of data is limited to about 12 GB for consistent operation. Data is refreshed periodically according to the Refresh Rate property.
● STATIC_OFF_HEAP	Data is pulled from an in-memory off heap store. This is the second fastest solution and the amount of data is limited only by the available RAM of the server. Data is refreshed periodically according to the Refresh Rate property.
Refresh Rate (seconds)	If the checkbox ‘auto’ is enabled, the Refresh Rate will be automatically set to 3,600 seconds (i.e. one hour) which is the default value. If the ‘auto’ box in unchecked, any value can be entered. The value is the number of seconds between a complete cache reload from the database. A lower value makes data more current but requires data to be loaded more often from the database. A higher value makes data less current but requires data to be loaded less often from the database. This value is not used for the Cache Storage Type NO_CACHE.
Use Secure Mode	If this checkbox is enabled, the web server will use secure connections (i.e. “https:”) rather than insecure connections (i.e. “http:”). If Secure Mode is selected, a Key Store Location and a Trust Store Location must be provided (see below).
Key Store Location	This property is only used if “Use Secure Mode” is enabled. This property holds the file path to a key store directory. The key store is a repository of secure certificates used for SSL encryption which, in turn, is a part of “https:”. The key store contains the application’s own certificate and private key.

Trust Store Location	This property is only used if “Use Secure Mode” is enabled. This property holds the file path to a trust store directory. The trust store contains a collection of certificate authority (CA) certificates trusted by the Ext Speeder application.

2.1.1.2 Dbms

The database instance that Ext Speeder is connected to will be represented as a Dbms node in the Project Tree. In these settings, many of the fields entered when first connecting to the database will be stored.

Property	**Meaning**
Dbms Type	Shows the selected database type (e.g. Oracle, MySQL, PostgreSQL or MariaDB). The number of available types, depends on which database drivers that are available to the tool at runtime. Usually, there is only one type available unless other types were added in the project’s POM file.
IP Address	If the checkbox ‘auto’ is enabled, the IP Address will be automatically derived as ‘127.0.0.1’ which is the localhost itself. If the ‘auto’ box is unchecked, any value like a custom ip address or an internet host name can be entered.
Port	If the checkbox ‘auto’ is enabled, the Port will be automatically derived as the default port for the selected Dbms Type (e.g. ‘3306’ for MySQL or ‘1521’ for Oracle). If the ‘auto’ box is unchecked, any value can be entered.
Username	If the checkbox ‘auto’ is enabled, the Username will be automatically derived as the default user for the selected Dbms Type (e.g. ‘root’ for MySQL). If the ‘auto’ box is unchecked, any integer value can be entered. If the Username is empty, no user name will be given to the database upon login.
Connection URL	If the checkbox ‘auto’ is enabled, the Connection URL will be automatically derived depending on the selected Dbms Type, IP Address and Port. If the ‘auto’ box is unchecked, any value can be added.

Note: For security reasons, Ext Speeder never saves a database password in any configuration files!

Tip: If a switch from a development database to one in production must be done, simply change the values for the **IP Address** and optionally the **Port**. These values can also easily be set programmatically in the Java code if needed.

2.1.1.3 Schema

For every schema that Ext Speeder is connected to there will be a Schema node in the Project Tree.

Property	**Meaning**
Exposed in REST	If this checkbox is enabled, all the underlying nodes like tables and columns can be exposed in the REST interface. If this box is unchecked, *all the underlying nodes will be hidden* from the REST interface, regardless of the underlying nodes REST exposure settings.

2.1.1.4 Table

For every table in the database, Ext Speeder can create a REST command that can be used to get a list of every row of that table. The default path of a table will consist of the Java Names of every node above the table in the tree.

Property	**Meaning**
Exposed in REST	If this checkbox is enabled, all the underlying nodes like columns can be exposed in the REST interface. If this box is unchecked, *all the underlying nodes will be hidden* from the REST interface, regardless of the underlying nodes REST exposure settings.
REST path	If the checkbox 'auto' is enabled, the REST path will be automatically derived as the Java Names of every node above the table in the tree (i.e. {Project Java Alias}.{Dbms Java Alias}.{Schema Java Alias}.{Table Java Alias}. If the 'auto' box is unchecked, any value can be entered.

Tip: If a custom REST path needs to be used, for an example to get something shorter, the **REST Path** property can be used.

2.1.1.5 Column

The columns from the database will make up the fields in JSON responses sent by the generated server application. Most of the settings in the Column node simply exist to reflect how the column works in the database (auto increment for an example), but some of them will affect the generated API.

Property	**Meaning**
Is Nullable	Indicates whether or not a value in this column can be set to null in the database. This should reflect how the database is configured.
JDBC Type to Java	Can be changed to control how the database data type for this column will be represented in the object oriented Java world. The selectable alternatives depends on which type the column has in the database.
REST path	If the checkbox 'auto' is enabled, the REST path will be automatically derived as the Java Names of every node above the table in the tree (i.e. {Project Java Alias}.{Dbms Java Alias}.{Schema Java Alias}.{Table Java Alias}. If the 'auto' box is unchecked, any value can be entered.
Is Auto Incrementing	Indicates whether or not a value in this column is generated automatically for new rows. This should reflect how the database is configured.
Exposed in REST	If this checkbox is enabled, the column can be exposed in the REST interface. If this box is unchecked, *the column will be hidden* from the REST interface.

Tip: If a column contains date, time or timestamp fields, setting the **JDBC Type to Java** property to something resulting in a `Long` or even `Integer` will save a lot of memory space.

2.1.1.6 Index

Indexes from the database will be examined when creating a column-oriented in-memory view of the data. This is a central part of the acceleration engine. If a single column index is set to **Unique**, Ext Speeder will assume that only one row in that column can have a certain value, which makes it possible to do optimizations based on that. Unique columns are much faster than non-unique ones.

2.1.1.7 IndexColumn

This node marks which column(s) that are indexed.

Note: If the name of a column in the database is changed, remember to also change the name of any IndexColumns referencing it in the graphical tool.

2.1.1.8 ForeignKey

Represents a foreign key in the database. These are used by Ext Speeder to organize related entities in the memory in a manner similar to graph databases.

2.1.1.9 ForeignKeyColumn

This node marks which column(s) that form the beginning of an edge.

Note: If a name of a column in the database is changed, remember to also change the name of any ForeignKeyColumns referencing it in the graphical tool.

2.1.1.10 PrimaryKeyColumn

This node marks which column(s) that are part of the primary key. The primary key is used as the unique identifier for an entity in its manager. Ext Speeder requires each table to have exactly one primary key column.

Note: If the name of a column in the database is changed, remember to also change the name of any PrimaryKeyColumns referencing it in the graphical tool.

2.1.2 Controlling the REST API

One of the most important things in the graphical tool is to configure how the REST API will be exposed. REST commands are created for every enabled table in the Project Tree. In this section the most common tasks, like hiding tables and renaming fields, using the graphical tool is explained.

2.1.2.1 Hide Table or Column in REST

When the graphical tool analyzes the database, it can not predict what information shall be exposed in the application. Therefore it is programmed to show every table and column possible. This is convenient when debugging a client application since the client has access to everything, but it is not recommended to have this setup in a production environment since it will allow anyone with access the API to read everything in the database!

To prevent a table or a column from being exposed in the REST API, select the node in the **Project Tree** and deselect the checkbox **Exposed in REST**.

Note: Disabling a node by unchecking **Enabled** means that it will not exist in the server application at all. Disabling **Exposed in REST** only hides it in REST queries, it can still be used from the Java code.

2.1.2.2 Change Name of Table or Column

When creating a backend for an existing client application, it is important to ensure that the names for something used in the API is the same as the table or column names used in the database. To change the name of a node in the REST API, uncheck **Auto** on the **REST Path** setting and write a custom path for that resource.

Tip: If the **auto**-setting is selected, the name will be determined using the **Java Alias** of every node above the column in the tree.

2.1.2.3 Change Value Type of Column

If a client applications expects a field to be formatted in a certain way, a TypeMapper can be used to translate between the database form and the exposed form. It might be that there is a VARCHAR column in the database with the values “Yes” and “No”, but we want the values to be sent as either “true” or “false”. This is a typical application for the TypeMapper feature.

To change mapping used for a particular column, select it in the **Project Tree** and select the type in the **JDBC Type to Java Type** dropdown. The following custom mappers exist in addition to the basic identity mappers:

- String to Local
- True/False to Boolean
- Yes/No to Boolean
- Date to Integer
- Date to Long
- Time to Integer
- Time to Long
- Timestamp to Integer
- Timestamp to Long
- Clob to String
- BigDecimal to Double

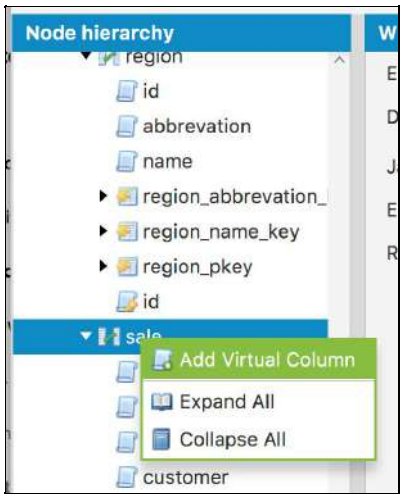
Tip: Representing various time units as Long or even Integer will reduce memory usage and increase performance. Mapping to Integer will produce a time representation in seconds which is very convenient to use on the front end.

Tip: Advanced users can write their own mappings and plug them into the graphical tool. More information about how to do this can be found in the **Ext Speeder Advanced Manual**.

2.1.2.4 Define Virtual Columns

A good practise when designing a database is to keep tables “normalized”, which means that tables should be small and concrete only covering a small concern per table. This is good for database design, but it is not necessarily what is expected in the client when sending a REST request. To bridge the gap between a normalized database and a denormalized client, there are something in Ext Speeder called **Virtual Columns**. A Virtual Column will look just like a normal column in the REST API, but in reality it is retrieved from another table in the background. Virtual Columns use foreign keys in the database to find information in other tables, but the retrieval is optimized using an in-memory cache on the server.

To create a virtual column in a particular table, Right-click on the Table node in the **Project Tree** and select **Add Virtual Column**.



A name will be assigned to the new column, but it is recommended to set a real name to whatever it shall be called in the API. Unlike ordinary columns, there is no **Database Name** for a virtual column since it does not really exist in the database. When the virtual column is created, it is possible to set which table and column it should virtualize. The items that are available in the dropdown-list are tables and columns that can be reached using one or several existing foreign keys. Make sure to pick a name that is unique amongst the table’s columns (i.e. a virtual column cannot have the same name as an existing column or another virtual column).

Example of a Virtual Column named “salesperson_country” that is using foreign keys from the “sale” table, via the “salesperson” and “office” tables to the “country” table where the column “name” is used. Thus, the Virtual Column will contain the name of the country where the office of the salesperson is located for the specific sale.

3 The Java API

Even if a lot of configuration can be done using the graphical tool, it is also possible to do changes in the generated code. This chapter in the manual is dedicated to those users that want to write some custom logic using the powerful features of the Ext Speeder server-side API. If no custom logic is needed, please skip directly to to **Chapter 4** for instructions on taking the application into production.

3.1 Navigating the Generated Code

The Ext Speeder class structure is quite straightforward. Source files are by default located in the `src/main/java/` folder and configuration files in the `src/main/json/` folder. The java sources are organized in packages named after the database structure.

The main entry point of the application is located directly in the folder named after the company and project name. The name of the file is **Main.java**.

```
src/main/java/com/company/myapp/Main.java
```

The main() method in the Main class just creates a helper class **StandaloneMainSupport** class and invokes its run() method. It is possible to override a number of methods in the **StandaloneMainSupport** class, for example the username() and password() methods. The generated logic for setting up the server and binding the various REST commands are located in **StandaloneMainSupport**'s superclass located at:

```
src/main/java/com/company/myapp/generated/GeneratedStandaloneMainSup
port.java
```

Below the directory of this class are the folders for the schemas and tables. For every table, a number of files will be created. A table file looks something like this:

```
src/main/java/com/company/myapp/db0/myapp/user/User.java
```

3.1.1 Entities and Managers

Every table in the database will result in a number of source files. Assuming that there is a table in the database called "user", then the following files will be created.

1. .../user/User.java
2. .../user/UserImpl.java
3. .../user/UserManager.java
4. .../user/UserManagerImpl.java

5. .../user/UserServlet.java
6. .../user/generated/GeneratedUser.java
7. .../user/generated/GeneratedUserImpl.java
8. .../user/generated/GeneratedUserManager.java
9. .../user/generated/GeneratedUserManagerImpl.java
10. .../user/generated/GeneratedUserServlet.java

The first package (1-4) is only generated once and contains a number of marker classes and interfaces. These are almost empty and will not be replaced if the code is regenerated. That means that custom code can be written in these files without the risk of the changes being overwritten. The second package called “generated” (5 - 8) is where the default implementation of these components are located. These files will be overwritten each time the code generator is called, so it is not advised to change them.

Here is a more detailed description of the purpose of each file:

1. **.../user/User.java**
   The interface of the user entity that will be exposed throughout the code. If new methods for a “user” entity needs to be exposed, this is where they go. User extends the `GeneratedUser` interface in the .../user/`generated`/ folder.

2. **.../user/UserImpl.java**
   The implementing class of the `User` interface. This class extends the `GeneratedUserImpl` class. If the implementation of a certain generated entity method needs to be changed, it can be overridden in this class.

3. **.../user/UserManager.java**
   The interface of the manager that handles operations on the User table. From this class, it is possible get a stream of users, persist entities to the database and so on. This extends the `GeneratedUserManager` interface. If new methods for a “user” manager needs to be exposed, this is where they go.

4. **.../user/UserManagerImpl.java**
   The implementing class of the manager interface. This extends the `GeneratedUserManagerImpl` class in the generated package. If the implementation of a certain generated manager method needs to be changed, it can be overridden here.

5. **.../user/UserServlet.java**
   This Servlet is used to serve HTTP requests from end users. It is used regardless if the application is run stand-alone or in a JavaEE server..

6. **.../user/generated/GeneratedUser.java**
   This interface is where all the generated methods for the user table is declared. It also contains fields for referencing the different user columns. More about this later on.

7. **.../user/generated/GeneratedUserImpl.java**
   This abstract class is where the standard implementation of the `GeneratedUser` interface is located. It is overridden by the `UserImpl` class.

8. **.../user/generated/GeneratedUserManager.java**
   The base interface of the manager.

9. **/user/generated/GeneratedUserManagerImpl.java**
   The abstract base implementation of the `GeneratedUserManager` interface. This is overridden by the `UserManagerImpl` class.

10. **/user/generated/GeneratedUserServlet.java**
    The abstract base implementation of the **UserServlet** servlet. This is overridden by the `GeneratedUserServlet` class.

3.2 Common Stream Operations

The Ext Speeder instance is built on top of the `Stream` interface introduced in Java 8. A stream is a pipeline of different actions that is put in front of a data source. When requesting an item from the pipeline, every action passes the request up the stream, applying it on the way down. Some operations can also be short-circuited since the structure of the whole pipeline is known before it is being executed. For example, just counting all items in a stream means that any sorting can be removed, since sorting does not influence the number of elements in the stream, it just modifies the order of the elements.

A stream from a particular table is obtained via its manager as shown hereunder:

```
final Stream<User> stream = users.stream();
```

Once a stream has been obtained, it is possible to add intermediate operations to it, until it is terminated. Thus, no elements will be retrieved until the stream is terminated. There is a number of different terminating operations available. The one used in the example simply executes the specified function once for each element in the stream, without guaranteeing any specific order.

```
users.stream().forEach(user -> {/* do something cool */});
```

The most common terminating operations are listed in the table below:

Terminating Operation	Function

`.forEach(consumer)`	Calls the specified consumer once for every entity without guaranteeing the order.
`.forEachOrdered(consumer)`	Calls the specified consumer once for every entity in order of appearance in the stream.
`.collect(collector)`	Collects the entities into a list, map, set, string or something completely else using a Collector.
`.count()`	Returns the number of elements in the stream.
`.findAny()`	Returns any single element from the stream. If there are several elements in the stream, any of them may be selected. If there are no elements in the stream, Optional.empty() is returned.
`.findFirst()`	Returns the first element in an ordered stream. If there are no elements in the stream, Optional.empty() is returned.

Simply iterating over an entire table is seldom useful. Instead, the usual way to go is to first filter out certain elements of interest. The standard way of doing this in Java 8 is to add one or several predicates using the `filter()`-method. A predicate is a function that takes some kind of element object and returns a `boolean`. If the returned boolean is true, the element "survives" the filter and propagates downstream. If the returned `boolean` is false, the element is dropped from the stream.

```
users.stream()
    .filter(user -> user.getId() < 1000)
    .forEach(user -> {/* do something cool */});
```

For performance reasons, Ext Speeder generates a number of optimized predicates that ***should be used***. These will utilize the in-memory storage structures to reduce the complexity of the search.

```
users.stream()
    .filter(User.ID.lessThan(1000))
    .filter(User.NAME.equalIgnoreCase("Lisa"))
    .forEach(user -> {/* do something cool */});
```

3.3 Formatting JSON Output

Often when writing REST applications, not every column needs to be included in the JSON output. In the User example used previously, we might have hashed passwords and other sensitive user information in the database that we don't want to expose. In the graphical tool, this is exclusions can be done automatically. This chapter provides a guide as to how Ext Speeder performs JSON formatting and how to modify the formatting.

3.3.1 Modifying the JSON Output

Entities are converted to JSON using the corresponding servlet method `toJson()`. If the JSON format needs to be changed programmatically, override this method in the main Servlet class. For example, if there is a “user” table, then override the `toJson(User entity)` method in the **UserServlet** class.

4 Deploying a Server Application

This chapter contains some guidelines on how to deploy the Ext Speeder application. An Ext Speeder application can be deployed in two ways:

1) Stand-alone. The application is started using its Main class.
2) In a JavaEE server like Tomcat. The application is deployed as a WAR file on the JavaEE server

4.1 Deploying Stand Alone

When deploying the application on a server that does not have an IDE or Maven installed, all the external JAR dependencies (e.g. the database driver) must be provided to the application upon start.

There is a convenient Maven plugin that can automatically pack the application together with all its dependencies in a composite "Über JAR". This JAR is a self-contained unit that can easily be deployed in a single step.

Read more on https://maven.apache.org/plugins/maven-shade-plugin/

4.2 Deploying in A JavaEE Server

To be able to deploy the application in a JavaEE Server, the **pom.xml** file first needs to be updated so that the element `<packing>jar</packing>` is changed to `<packing>war</packing>` instead. When the pom.xml file has been changed, rebuild your project and a war file will be created. This war file contains everything needed to be deployed in a JavaEE server.

Some of the files that are being generated will control how you application will be deployed. Make sure to review the file web.xml to understand how you server resources are mapped.

When the application is deployed, there is a start page located at the deployment path named **index.jsp**. Access this page and enter the credentials for database access and press the button "Initialize Ext Speeder". The Ext Speeder application cannot serve requests unless it has been initialized properly.

There is also a shutdown servlet that will clean up all resources used by Ext Speeder. The servlet is located at the deployment path and is named "shutdown". Make sure you shutdown Ext Speeder before you undeploy you Ext Speeder application.

4.3 Setting JVM parameters

Since Ext Speeder stores all the selected data in-memory, the JVM/JavaEE server must reserve sufficient memory upon application start. The default memory size is usually 2 GByte. If the application runs out of memory, increase the available memory space allocation. For example, by providing the following command line parameters:

```
java -Xms8G -Xmx8G -jar MyApp.jar
```

This will allocate 8 GByte of memory to the JVM.

5 Requirements and Performance

5.1 Database types

Ext Speeder supports the following database versions:

Database type	Version	JDBC version
Oracle	12.1	ojdbc7, 12.1.0.1.0
MySQL	5.7.9	5.1.38
PostgreSQL	9.4.5	9.4-1206-jdbc4
MariaDB	10.1	1.4.0

The oracle database JDBC driver must be downloaded and installed manually.

5.2 JVM types

Ext Speeder supports the following JVMs:

Jvm type	Version	Revision
Oracle JDK	8	>=40
Open JDK	8	>=40

5.3 JavaEE Servers

Ext Speeder supports the following JavaEE servers:

JavaEE server	Version	Comment
Tomcat	8.0.27	Java EE 7, Servlet 3.1
Glassfish	4.1.1	Java EE 7, Servlet 3.1

If you use Glassfish, you need to setup database access through “Driver” access. Ext Speeder might work with any JavaEE server that supports Servlet 3.1.

5.4 Data Sizes

Ext Speeder handles the following data sizes:

Restriction	Amount	Comment
Elements total per JVM	100,000,000	Elements = Σ rows*columns
Rows per table	25,000,000	
Columns per table	300	

These are the maximum supported data sizes. Ext Speeder might however continue to work with larger data sets, but potentially with reduced performance.

The number of elements are simply the number of rows times the number of columns, for every selected table.

5.5 Primary Key Requirement

Because Ext Speeder keeps an in-memory representation of all rows, the rows must be uniquely identifiable. This is made using the primary key. Ext Speeder handles one primary key. This means that all tables that Ext Speeder is handling, must have exactly one primary key per table.

If a primary key is not available or if there are several primary keys defined for a table, the developer can, via the JSON configuration file, set any column to a virtual primary key. This will then work as an auxiliary primary key column, provided that the column has unique non-null values throughout the table.

5.6 Static data

Upon application start, Ext Speeder will read the entire database into memory and order the content in its internal format. In order to refresh the in-memory data set, the application has to be restarted.

5.7 Performance

Ext Speeder is designed with performance in mind. Even if the amount of data is increased by say 200%, the latency for queries will only increase marginally (by perhaps only 10%).

Unique columns will give better performance than non-unique columns.

Ext Speeder is designed to be massively parallel and with low coupling between threads. This means that the more CPU cores available to the application, the more capacity the application will have (i.e. increasing the number of CPU cores from four to eight means that the application can serve almost twice as many request per second).

The startup time for a Ext Speeder application depends on several factors but is approximately proportional to the amount of data loaded.

5.8 Benchmarks

For applications with tens of million elements, Ext Speeder can typically serve thousands of queries per second and with latencies typically in milliseconds.

5.9 Query Planning

When an Ext Speeder REST request is received, the query is analysed and optimized to give the best predictable performance possible. There are four basic components in a query:

- Filter - Which rows to include in a query
- Sort - The order of rows
- Skip - How many initial rows shall be skipped before rows are outputted
- Limit - How many of the filtered, sorted and non-skipped rows to include

Due to memory efficiency constraints, Ext Speeder cannot predict all possible combinations of the four basic components at runtime. Keep the queries as simple as possible for best performance and try to avoid filters in performance sensitive applications (filters are the hardest to predict). Limit does not contribute to query complexity and should be used whenever possible.

Appendix A

List of optimized predicates:

Operator	**JSON**	**Comment**
Equals	eq	Optimized
Not equals	ne	Optimized
Less than	lt	Optimized
Less or equal	le	Optimized
Greater than	gt	Optimized
Greater or equals	ge	Optimized
Contains	like	O(n) behavior

N.B. The operators 'in', 'notin' and 'regexp’ are not supported.