# General Project Setup

To generate the persistence layer using HotRod you need a live database, the HotRod library (hotrod-<version>.jar file) and the HotRod Configuration File.

HotRod reads the Configuration file, then retrieves metadata from the live database, and produces the DAO Java classes. The MyBatis Generator Overview and Spring JDBC Generators sections describe the file locations and directory structure in detail.

## MyBatis Generator Directory Setup

In sum, the MyBatis Generator requires the following file to run:

* The HotRod Library for example at: lib/hotrod-3.0.0.jar
* The HotRod Runtime Library for example at: lib/hotrod-runtime-3.0.0.jar
* The hotrod Ant Task included in the build.xml file
* HotRod Configuration File hotrod.xml
* MyBatis Configuration file template mybatis-template.xml
* The SessionFactory java/sessionfactory/DatabaseSessionFactory.java

Once it's run HotRod generates the following files. For simplicity we'll assume there's a <table> tag for the table BOOK and the database is H2:

* DAO Java classes
* DAO Primitives Java classes
* Mappers
* MyBatis Configuration File

The directory structure for a stand alone (command-line) program can look like:

project/

+- lib

+- hotrod-3.0.0.jar

+- hotrod-runtime-3.0.0.jar

+- h2-1.3.176.jar

build.xml

hotrod.xml

mybatis-template.xml

+- src/

+- java/

+- sessionfactory/

+- DatabaseSessionFactory.java

+- auto-generated/

+- java/

+- primitives/

**+- BookDAOPrimitives.java**

**+- BookDAO.java**

+- mappers/

+- primitives/

**+- mybatis-configuration.java**

**+- primitives-book-dao.java**

In this example all the initial directories and libraries need to be put in place by the developer. HotRod generates all the files in the auto-generated directory.

Now, a JEE web project would place the file in slightly different places. Below is a proposed structure for a JEE web project:

project/

+- lib

+- hotrod-3.0.0.jar

build.xml

hotrod.xml

mybatis-template.xml

+- src/

+- java/

+- sessionfactory/

+- DatabaseSessionFactory.java

+- auto-generated/

+- java/

+- primitives/

**+- BookDAOPrimitives.java**

**+- BookDAO.java**

+- WebContent/

+- WEB-INF/

+- classes

+- primitives/

**+- mybatis-configuration.java**

**+- primitives-book-dao.java**

+- lib

+- hotrod-runtime-3.0.0.jar

+- h2-1.3.176.jar

This setup presents you a default directory structure that can be suitable to develop a stand alone and JEE web application.

## Ant Task

HotRod runs as a Apache Ant task, that can be called from Eclipse, or from the command-line if you ever happen to need to run it on a headless server with no GUI.

In order to run the HotRod's Ant task you first need to define it as in:

<taskdef name="hotrod" classname="org.hotrod.ant.HotRodAntTask">

<classpath>

<pathelement location="lib/hotrod-3.0.0.jar" />

<pathelement location="lib/h2-1.3.176.jar" />

</classpath>

</taskdef>

Once the <hotrod> Ant task is defined, you can use it in the Ant script as in:

<hotrod url="jdbc:h2:tcp://localhost:12345/db001"

driverclass="org.h2.Driver"

username="sa"

password=""

catalog="DB001"

schema="PUBLIC"

generator="MyBatis"

facets=""

configfile="hotrod.xml"

display="list" />

The parameters shown below specify runtime details such as the live database location and credentials, the default catalog and schema, the configuration file location and the facets to consider.

You can see a running example of this in the Hello World example for MyBatis or Spring JDBC.

## HotRod Runtime Library

When running the application that uses DAOs Java classes generated by HotRod, the application needs to use the HotRod Runtime Library (hotrod-runtime-<version>.jar) to perform certain operations. This is a very small library with helper routines that simplify the DAO generation. The library jar file includes the runtime source code should you need to inspect it.