

## Getting started with the embedded edition

This step-by-step manual guides you through the set up of a simple Java program with which the embedded edition of Match4J can be operated.

If you have additional questions or encounter any problems, go to our website [www.match4j.com](http://www.match4j.com).

How Match4J works will be clarified by developing a simple Java application. This application will make use of the example configuration “*examples/simple/SimpleExample.conf*” and the additionally supplied test data.

This example defines a car database in which can be searched for the most ideal car.

### Step 1: Checking that Match4J is installed properly

Before being able to work with the match engine, it is important that it is installed properly. Consult the installation manual to check the installation process.

### Step 2: Creating a new Java program

Create a Java class named “*GettingStartedExample*”, which only contains a “*main*” method, and save this file as “*GettingStartedExample.java*”. The program should read the name of the configuration file from the first parameter.

```
public class GettingStartedExample {
    public static void main(String[] args) throws Exception {
        String configPath = args[0];
    }
}
```

### Step 3: Importing the required packages

The public interfaces reside in the “*com.match4j*” package. Add this package to the import list of the “*GettingStartedExample*” class. Also, add some other imports that will be needed in the following steps.

```
import com.match4j.*;
import java.io.FileReader;
import java.util.Iterator;

public class GettingStartedExample {
    ...
}
```

### Step 4: Initializing the match engine

All communication with the match engine occurs via the “*IMatchEngine*” interface. This is obtained via a factory.

```
public static void main(String[] args) {
    ...
    MatchEngineFactory factory =
        MatchEngineFactory.newInstance();
    IMatchEngine matchEngine =
        factory.createMatchEngine(
            new FileReader(configPath));
}
```

## Getting started with the embedded edition

### Step 5: Defining the ideal result

In this step, you define the ideal car. You can, for instance, enter the model and maximum price on the command line.

```
public static void main(String[] args) {
    ...
    IEntity type =
        matchEngine.getEntityTypeById("Car");

    IVolatileEntity idealCar =
        matchEngine.createVolatileEntity(type);

    idealCar.updateAttribute("brand", args[1]);
    idealCar.updateAttribute("sunroof", new Boolean(args[2]));
    idealCar.updateAttribute("doors", new Integer(args[3]));
}
```

First, the type definition of the entity “*Car*” is asked for. An entity type defines which attributes an entity contains. Then, a new entity is created which represents your most ideal car. It is a “*volatile*” entity, because it only exists as an object and will not be stored in a database. Finally, the properties of this new entity will be set.

The match engine will eventually search for cars that resemble the car with the assigned model, sunroof and number of doors.

### Step 6: Creating the match request

In this step, you will build a so-called *match request*. It contains instructions on how Match4J has to perform a search.

```
public static void main(String[] args) {
    ...
    IMatchRequest request = matchEngine.createMatchRequest();
    request.setMaxMatchTime(1000);
    request.setSufficientMatchPercentage(10.0f);
    request.setMaxResultListLength(10);

    IMatchRuleList matchRuleList =
        matchEngine.createMatchRuleListFromTemplate("Car2Car");

    request.setProfile(idealCar);
    request.setMatchRules(matchRuleList);
}
```

First, a match request is made. Then, the conditions are set that the match engine should not search for longer than 1000 ms, that the matching engine does not return a result of more than 10 cars, and that in the top ten contains only cars that resemble your ideal car for at least 10%.

## Getting started with the embedded edition

Furthermore, a collection of match rules is taken from the configuration. The match rules determine how your ideal car should be matched to other cars. In other words, the match rules define the semantics of “two cars should resemble each other”.

Finally, the car entity and the match rules are passed to the match request.

### Step 7: Executing the request

The complete request is build and can be executed by the match engine. After the match engine has executed this request, the result will be stored in an “*IMatchResultList*” object.

```
public static void main(String[] args) {
    ...
    IMatchResultList result = matchEngine.execute(request);
}
```

### Step 8: Viewing the match result

This final step consists of viewing the match result.

```
public static void main(String[] args) {
    ...
    StringBuffer resultText = new StringBuffer();
    for (Iterator iter = result.iterator(); iter.hasNext();) {
        IMatchResultElement resultElement =
            (IMatchResultElement) iter.next();

        IEntity entity = resultElement.getEntity();
        resultText.append(" Entity ");
        resultText.append(entity.getId());
        resultText.append(" --> totalscore ");
        resultText.append(resultElement.getTotalScore());
        resultText.append("\n");
    };
    System.out.println(resultText.toString());
}
```

### Step 9: Compiling the example

After having followed all the steps, you can compile the program with the following elements in the classpath:

- match4j/lib/match4j.jar
- match4j/lib/log4j-1.2.7.jar

The example below shows how to compile the program using a Sun Java compiler on the Windows platform:

```
javac -classpath
    match4j\lib\match4j.jar;match4j\lib\log4j-1.2.7.jar
    GettingStartedExample
```

## Getting started with the embedded edition

The example compile command expects that in the directory in which the test program is stored, the match engine is located in a directory match4j.

### Step 10: Running the example

When the program compiled successfully, you can run the program.

- match4j/lib/match4j.jar
- match4j/lib/log4j-1.2.7.jar
- match4j/conf
- match4j/examples/simple
- .

The example below shows how to start the testprogram using a Sun JRE on the Windows platform:

```
java
  -cp match4j\lib\match4j.jar;
    match4j\lib\log4j-1.2.7.jar;
    match4j\examples\simple;
    match4j\conf;
    .

  -Dlog4j.configuration=log4j_console.conf

  GettingStartedExample

  match4j\examples\simple\SimpleExample.conf Ford true 3
```

The above given example expects that in the directory in which the test program is stored, the match engine is located in a directory match4j. Additionally, in the example, the ideal car is a Ford with a sun roof and three doors.