

## Getting started with the server edition

This step-by-step manual guides you through the set up of a simple test program for the Match4J server and provides an overview of the request / response XML.

In order to be able to understand this manual, no previous knowledge of programming languages is required. However, knowledge of XML and DTDs is expected.

If, after having read the manual, you have additional questions, go to our website [www.match4j.com](http://www.match4j.com).

### Basic ideas of the architecture and functioning of the Match4J server

The server edition of Match4J has a client/server architecture which is similar to that of most web servers. The Match4J server is a Java application that listens for match requests on a TCP port.

When clients have established a connection, they are required to send the match request as XML in one line followed by an ENTER. When the match4J server comes across an ENTER, the incoming message will be parsed and executed in the built-in match engine.

After the match request has been processed, the server will send back an answer to the client in XML and will automatically disconnect.

The XML of the answer complies with the DTD as described in the file *“documents/config/matchresult.dtd”*. The XML of the request should comply with the DTD as described in *“documents/config/matchrequest.dtd”*.

### What are we going to create?

The goal of this step-by-step manual is to help you execute a working example. For this purpose, the example in the folder *“examples/simple”* is used.

### 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: Starting the Match4J server

The Match4J server has to be started before match requests can be executed. After being started, the match server continuously listens for clients on a TCP port. This port is configured as *“1234”* in the startup scripts.

First, start the server:

1. go to the *“examples/simple”* directory
2. start the *“runserver.bat”* or *“runserver.sh”* script. The server will now be started and all logging information will be send to the console.

### Step 3: Viewing the example match request

In the directory *“examples/simple”*, there is a file *“matchrequest.xml”*, which describes a complete match request.

## Getting started with the server edition

### Step 4: Sending the request to the Match4J server

In order to send the request to the server, you will need to do the following:

1. open the “*matchrequest.xml*” document and copy it to the clipboard
2. establish a connection (for instance, with telnet) to the server on port 1234, for example, “*telnet localhost 1234*”
3. paste the XML text from the clipboard and press ENTER

Now the server will process the request and send back the result to the client and disconnect.

### Step 5: Understanding the answer

The answer the Match4J server returns, complies with the DTD as described in “*documents/config/matchresult.dtd*”.

The root tag of the match result is always “*match4jresult*”. Inside this tag, the match engine returns either a list of matching entities (tag “*matchresultlist*”) or an error message (tag “*error*”).

The example below shows an error message, which consists of an error *code*, a *description* and an optional *details* tag:

```
<match4jresult>
  <error>
    <code>xml</code>
    <description>
      Error constructing the request because of
      an XML problem
    </description>
    <details>
      End tag expected for element 'employee'
      on line 10
    </details>
  </error>
</match4jresult>
```

The following error codes can occur:

- *xml*: this message occurs when the XML is invalid. Make sure the XML request complies with the *matchrequest.dtd*.
- *enginecore*: the engine returns this error if the XML is valid, but contains invalid data. For example a reference to a non-existing entity type definition.
- *timeout*: the match request specifies a timeout (ms). The engine returns this error code when the timeout is reached before the request was processed.
- *unknown*: this message is returned if something else failed while processing the XML request.
- *engineexception*: finally, the match engine returns this error code when something failed during matching. For example a broken database connection.

## Getting started with the server edition

The example below shows the result of a successful match. The answer contains two matching entities (*entity1* matches for 85% and *entity2* matches for 70%).

```
<match4jresult>
  <matchresultlist>
    <matchresultelement>
      <entityid>entity1</entityid>
      <totalscore>85.0</totalscore>
      <matchrulescores>...</matchrulescores>
    </matchresultelement>

    <matchresultelement>
      <entityid>entity2</entityid>
      <totalscore>70.0</totalscore>
      <matchrulescores>...</matchrulescores>
    </matchresultelement>
  </matchresultlist>
</match4jresult>
```

The example shows the total scores for each entity. The match engine also supplies information on how this score was calculated, by returning the score for each match rule:

```
<matchrulescores>
  <matchrulescore>
    <matchruleid>matching_age</matchruleid>
    <score>90</score>
  </matchrulescore>

  <matchrulescore>
    <matchruleid>matching_education</matchruleid>
    <score>80</score>
  </matchrulescore>
</matchrulescores>
```

## Getting started with the server edition

### Understanding the request

The match request should comply with the DTD as described in “*documents/config/matchrequest.dtd*”. The example below shows a basic match request:

```
<matchrequest>
  <maxmatchtime>2000</maxmatchtime>
  <sufficientmatchpercentage>80</sufficientmatchpercentage>
  <maxresultlistlength>10</maxresultlistlength>
  <haltonfullresultlist>true</haltonfullresultlist>

  <matchrulelist>...</matchrulelist>
  <profile>...</profile>
</matchrequest>
```

- *maxmatchtime*: the maximum number of ms the match engine is allowed to take for executing the request. If the timeout expired before the match engine was able to start the request, the match engine returns a “*timeout*” error. However, if the match engine did start executing the request, then the engine returns the matched entities found so far.
- *sufficientmatchpercentage*: a percentage between 0 en 100 that guarantees that entities with a match score below this percentage will not be included in the match result.
- *maxresultlistlength*: defines the maximum number of entities in the result list.
- *haltonfullresultlist*: indicates if the match engine should iterate over all entities. This value can be “*true*” or “*false*”. When “*false*”, the match engine returns the result when it finds “*maxresultlistlength*” entities with a score of at least “*sufficientmatchpercentage*”. For optimal performance, this value should be “*false*” and for optimal match quality, it should be “*true*”.

Next, we have to specify which match rules to use. The match rules and match rule lists should be pre-defined in the match engine configuration. In the simplest form, you can specify which match rule set to use:

```
<matchrulelist>
  <id>employee2job</id>
</matchrulelist>
```

This example used the pre-configured matching configuration. However, it is possible to override some of the configuration settings. For example, you can disable match rules or override their weight, minimal match percentage and match direction.

## Getting started with the server edition

The example below overrides the settings of two match rules. With the empty “*disabled*” tag it is possible to completely disable a match rule. The second match rule overrides it’s parameters.

The direction parameter can have the following values: “*PROFILE\_TO\_ENTITY*”, “*ENTITY\_TO\_PROFILE*” and “*TWO\_WAY*”. Please consult the configuration manual for a detailed overview.

```
<matchrulelist>
  <id>employee2job</id>

  <matchrule>
    <id>age_matching</id>
    <disabled/>
  </matchrule>

  <matchrule>
    <id>education_matching</id>
    <weight>0.5</weight>
    <requiredmatchpercentage>75</requiredmatchpercentage>
    <direction>PROFILE_TO_ENTITY</direction>
  </matchrule>
</matchrulelist>
```

Finally, you have to specify the profile entity. This is the entity you want to match with all the entities in the data cache.

```
<profile>
  <entitytypeid>employee</entitytypeid>

  <attributes>
    <attribute>
      <id>age</id>
      <value>30</value>
    </attribute>

    <attribute>
      <id>education</id>
      <value>high</value>
    </attribute>
  </attributes>
</profile>
```

The example above shows a profile definition, that starts with the definition of the entity type (in this case “*employee*”). Next, you have to specify zero or more attributes, embedded in the “*attributes*” tag). Make sure the id’s are defined in the configuration file and that the values match the configured types.