MATCH4J

1

Match rules

Match rules define the way that entities are matched to each other.

This document describes the match rules that are supplied by the Match4J package.

Furthermore, this document describes how you can write your own custom match rules.

Match rules are configured in de Match4J configuration file. Please consult the configuration tutorial (match4j_configuration.pdf) on how to configure the general match rule properties. This document describes the configuration in more detail.

The Match4J package contains three match rule implementations:

- SimpleObjectMatchRule
- StringMatchRule (new in 1.1)
- RangeMatchRule (new in 1.1)
- StringSetMatchRule new in 1.1)
- GlidingScaleMatchRule
- MatrixMatchRule
- MatrixStringSetMatchRule (new in 1.1)

SimpleObjectMatchRule

The simplest match rule implementation compares an attribute of two entities to each other. What is important is that the type of both the attributes are equal (for example, you cannot match an integer attribute with a boolean attribute).

This match rule is implemented by the class
"com.match4j.matchrules.SimpleObjectMatchRule"

There are two required parameters: *"RequestAttributeId"* and *"ResultAttributeId"*. They define the attributes of the request entity and of the result entity, respectively.

The following example shows the configuration of a match rule, where the *"sunroof"* attributes are compared to one another.

```
<implementingclass>
        <classname>
                com.match4j.matchrules.SimpleObjectMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>sunroof</value>
        </parameter>
        <parameter>
                <key>ResultAttributeId</key>
                <value>sunroof</value>
        </parameter>
</implementingclass>
```

2

Match rules

StringMatchRule

This match rule implementation can be used to match the occurence of a text string. When one or more occurences are found, the score is 100%. Otherwise the score is 0%. The match rule is case sensitive. For example:

"Java" matches with *"Experienced Java-programmer"*
"Java" does not match with *"dotnet"*
"Java" does not match with *"JAVA"*

This match rule is implemented by the class
"com.match4j.matchrules.StringMatchRule"

There are two required parameters: *"RequestAttributeId"* and *"ResultAttributeId"*. They define the attributes of the request entity and of the result entity, respectively. Only String type attributes are allowed.

The following example shows the configuration of a match rule, where the *"brand"* attributes are compared to one another.

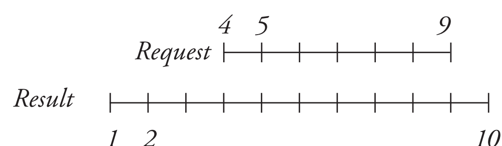
```
<implementingclass>
        <classname>
                com.match4j.matchrules.StringMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>brand</value>
        </parameter>
        <parameter>
                <key>ResultAttributeId</key>
                <value>brand</value>
        </parameter>
</implementingclass>
```

RangeMatchRule

This match rule implementation can be used to:
a) match a point on a range,
b) match a range on a point and
c) match two ranges.

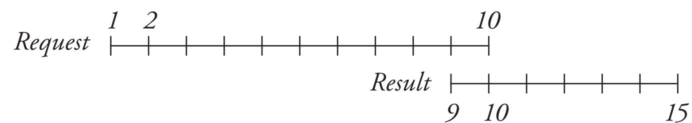
When matching a point and a range, the score is 100% if the point intersects with the range. Otherwise the score is 0%.

When matching two ranges, the score is 100% if the request range is completely contained in the result range:



Match rules

Otherwise, the score is a percentage, defined by how much the result range intersects with the request range. For example, suppose you want to find something in the range of 1 - 10. When matched to an entity that defines a range of 9 - 15, then the match is 10%.



This match rule is implemented by the class
"com.match4j.matchrules.RangeMatchRule"

There are four required parameters: *"RequestLowValueAttributeId"*, *"RequestHighValueAttributeId"*, *"ResultLowValueAttributeId" and "ResultHighValueAttributeId"*. The first two parameters define request entity attributes that contain the low and high range boundaries and the third and fourth parameters define result entity attributes that contain the low and high range boundaries.

Only number attributes are allowed (Short, Integer, Long, Float and Double).

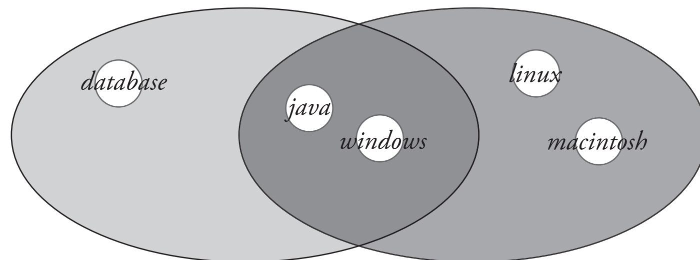
The following example shows the configuration of a match rule:

```
<implementingclass>
        <classname>
                com.match4j.matchrules.RangeMatchRule
        </classname>
        <parameter>
                <key>RequestLowValueAttributeId</key>
                <value>weight_min</value>
        </parameter>
        <parameter>
                <key>RequestHighValueAttributeId</key>
                <value>weight_max</value>
        </parameter>
        <parameter>
                <key>ResultLowValueAttributeId</key>
                <value>weight_min</value>
        </parameter>
        <parameter>
                <key>ResultHighValueAttributeId</key>
                <value>weight_max</value>
        </parameter>
</implementingclass>
```

Match rules

StringSetMatchRule

This match rule implementation matches two string sets with each other. A string set is defined as a text string with the elements separated using a delimiter.
For example the string *"database;java;windows;linux;macintosh"* defines a string set that consists of *"database"*, *"java"*, *"windows"*, *"linux"* and *"macintosh"*. The picture below shows two entities. The entity on the left contains a set of *"database"*, *"java"* and *"windows"* and the entity on the right defines a set of *"java"*, *"windows"*, *"linux"* and *"macintosh"*. Two out of five elements overlap, which results in a score of 2/5 = 40%.



This match rule is implemented by the class
"com.match4j.matchrules.StringSetMatchRule"

There are tree required parameters: *"RequestAttributeId"*, *"ResultAttributeId"* and *"Delimiter"*. They define the attributes of the request entity, the result entity and the delimiter, respectively. Only String type attributes are allowed.

The example below shows the configuration of a match rule, where the *"brands"* attributes are compared to one another.

```
<implementingclass>
      <classname>
            com.match4j.matchrules.StringSetMatchRule
      </classname>
      <parameter>
            <key>RequestAttributeId</key>
            <value>brands</value>
      </parameter>
      <parameter>
            <key>ResultAttributeId</key>
            <value>brands</value>
      </parameter>
      <parameter>
            <key>Delimiter</key>
            <value>;</value>
      </parameter>
</implementingclass>
```

Match rules

Gliding scale match rule

In real time situations it often happens that a match rule is not exactly matched. For example some one is looking for a partner with the age 25. Is a person with the age of 24 or 26 not suited ? With gliding scale match rules it can be determined that the ideal age is 25, but that 26 matches for 90%.

This is a very powerful method because it will also find results that do not match for 100% but are still acceptable results. Without gliding scale technology a lot of potential results will be missed.

The gliding scale match rule is implemented in the class *"com.match4j.matchrules.GlidingScaleMatchRule"*.

To use this match rule, it has to be configured in the Match4J config file:

```
<matchrule>
    ...
    <implementingclass>
        <classname>
            com.match4j.matchrules.GlidingScaleMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>ageofpartner</value>
        </parameter>
        <parameter>
            <key>ResultAttributeId</key>
            <value>age</value>
        </parameter>
        <parameter>
            <key>GlidingScaleConfig</key>
            <value>ageglidingscale.conf</value>
        </parameter>
    </implementingclass>
</matchrule>
```

There are three required parameters: *"RequestAttributeId"* and *"ResultAttributeId"*. They define the attributes of the request entity and of the result entity, respectively and a *"GlidingScaleConfig"* parameter that refers to a configuration file that contains detailed gliding scale configuration.

The gliding scale configuration file is an XML document that should comply to the DTD as defined in the *"document/config/glidingscale-config.dtd"*.

The example below shows the definition of a gliding scale match rule that can be used to match the age of persons.

Match rules

```
<glidingscale>
        <id>AgeGlidingScale</id>
        <type>relative</type>
        <controlpoints>
                <controlpoint>                              <!-- 1 -->
                        <deviation>0</deviation>
                        <percentage>100</percentage>
                </controlpoint>
                <controlpoint>                              <!-- 2 -->
                        <deviation>20</deviation>
                        <percentage>0</percentage>
                </controlpoint>
                <controlpoint>                              <!-- 3 -->
                        <deviation>-10</deviation>
                        <percentage>0</percentage>
                </controlpoint>
        </controlpoints>
</glidingscale>
```

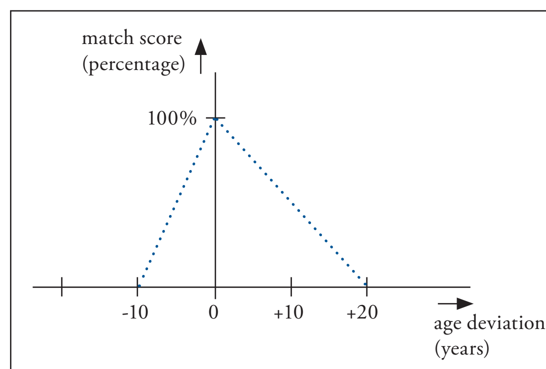
The gliding scale is of the type *"relative"*, which means that the deviation values are interpreted as percentages. The gliding scale consists of *"control points"*. A control point consists of a deviation and a percentage. In this example there are three control points.

Controlpoint 1:
- the 'top' of the gliding scale,
- values with 0 percent deviation match for 100 percent.

Controlpoint 2:
- values that deviate for +20 percent or more match for 0 percent.

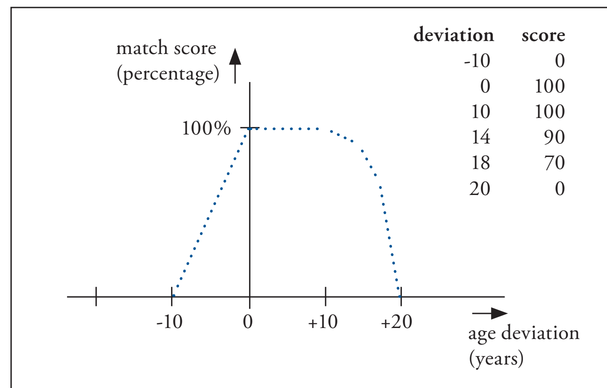
Controlpoint 3:
- values that deviate for -10 percent or more match for 0 percent.

The image below shows this gliding scale configuration in a graph:

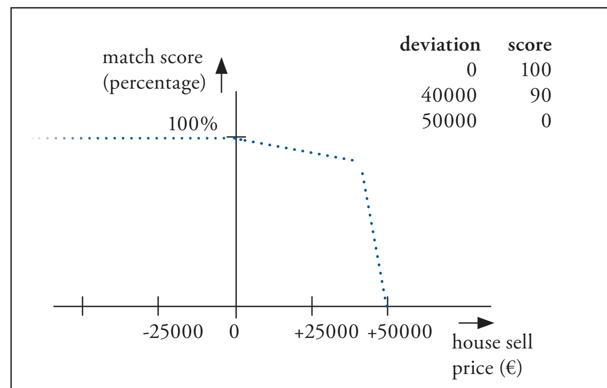


Match rules

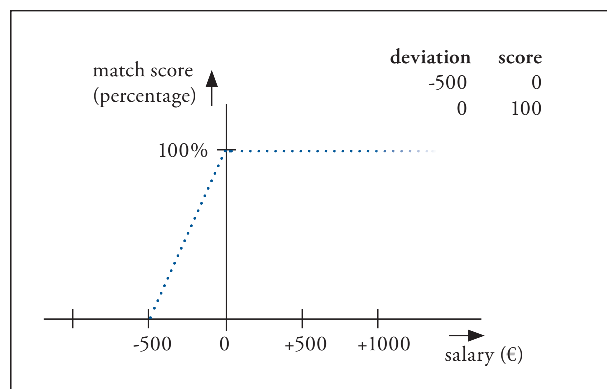
Here are some more examples:



This is almost the same example as before, now with more control points.



The example above shows a gliding scale configuration for matching a suitable house. A house that is at or below the requested price matches 100%. Houses that are € 40000 above the requested price still match for 90%. Houses that are € 50000 or more have a match score of 0%.



Match rules

This examples defines a gliding scale match rule for salary matching. When more than the requested salary is offered, this rule matches 100%. When € 500 or more is offered below the requested salary, this match rule scores 0%.

MatrixMatchRule

For a lot of discrete domains, matrices can be used. For example *"education"*. *"college"* level is requested, but "*high school"* may match for 75%.

Using matrices, all combination of discrete values can be matched. The example below shows how education level values match:

		Matched data		
		low	middle	high
Requested	low	100%	50%	10%
	middle	50%	100%	50%
	high	10%	50%	100%

For example, when a *"medium"* education level is requested, then a *"low"* education level is defined to match 50% and a *"middle"* education level as 100% (naturally) and a *"high"* education level as 50%.

The matrix match rule is implemented in the class *"com.match4j.matchrules.MatrixMatchRule"*.

To use this match rule, it has to be configured in the Match4J config file:

```
<matchrule>
    ...
    <implementingclass>
        <classname>
            com.match4j.matchrules.MatrixMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>education</value>
        </parameter>
        <parameter>
            <key>ResultAttributeId</key>
            <value>education</value>
        </parameter>
        <parameter>
            <key>MatrixConfig</key>
            <value>educationmatrix.conf</value>
        </parameter>
    </implementingclass>
</matchrule>
```

There are three required parameters: *"RequestAttributeId"* and *"ResultAttributeId"*. They define the attributes of the request entity and of the result entity, respectively and a *"MatrixConfig"* parameter that refers to a configuration file that contains detailed matrix configuration.

9

Match rules

The matrix match configuration file is an XML document that should comply to the DTD as defined in the *"document/config/matrix-config.dtd"*.

The example below shows the definition of a matrix match rule that can be used to match the education of persons.

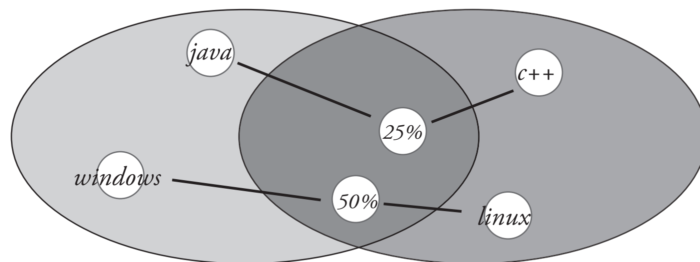
```
<matrix>
	<matrixid>EducationMatrix</matrixid>
	<fieldtype>java.lang.String</fieldtype>
	<fields>
		<field>
			<profilevalue>low</profilevalue>
			<entityvalue>low</entityvalue>
			<percentage>100</percentage>
		</field>
		<field>
			<profilevalue>low</profilevalue>
			<entityvalue>middle</entityvalue>
			<percentage>50</percentage>
		</field>
		<field>
			<profilevalue>low</profilevalue>
			<entityvalue>high</entityvalue>
			<percentage>10</percentage>
		</field>


		<field>
			<profilevalue>middle</profilevalue>
			<entityvalue>low</entityvalue>
			<percentage>50</percentage>
		</field>
		<field>
			<profilevalue>middle</profilevalue>
			<entityvalue>middle</entityvalue>
			<percentage>100</percentage>
		</field>
		<field>
			<profilevalue>middle</profilevalue>
			<entityvalue>high</entityvalue>
			<percentage>50</percentage>
		</field>


		<field>
			<profilevalue>high</profilevalue>
			<entityvalue>low</entityvalue>
			<percentage>10</percentage>
		</field>
		<field>
			<profilevalue>high</profilevalue>
			<entityvalue>middle</entityvalue>
			<percentage>50</percentage>
		</field>
		<field>
			<profilevalue>high</profilevalue>
			<entityvalue>high</entityvalue>
			<percentage>100</percentage>
		</field>
	</fields>
</matrix>
```

This is all you need to configure for the exucation example given earlier in this chapter.

Match rules

MatrixStringSetMatchRule

This match rule combines the *"matrix"* match rule and the *"string set"* match rule. This match rule works exactly like the string set match rule, but for each element in the set, it now uses the matrix to calculate the match.



		Matched data			
		java	c++	windows	linux
Requested	java	100%	25%	0%	0%
	c++	25%	100%	0%	0%
	windows	0%	0%	100%	50%
	linux	0%	0%	50%	100%

The matrix match rule is implemented in the class *"com.match4j.matchrules.MatrixStringSetMatchRule".*

To use this match rule, it has to be configured in the Match4J config file:

```
<matchrule>
    ...
    <implementingclass>
        <classname>
            com.match4j.matchrules.MatrixStringSetMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>education</value>
        </parameter>
        <parameter>
            <key>ResultAttributeId</key>
            <value>education</value>
        </parameter>
        <parameter>
            <key>Delimiter</key>
            <value>;</value>
        </parameter>
        <parameter>
            <key>MatrixConfig</key>
            <value>educationmatrix.conf</value>
        </parameter>
    </implementingclass>
</matchrule>
```

Match rules

There are four required parameters. The *"RequestAttributeId"* and *"ResultAttributeId"* define the attributes of the request entity and of the result entity. The *"Delimiter"* parameter defines the set-delimiter. Finally, the *"MatrixConfig"* parameter refers to a configuration file that contains the matrix configuration.

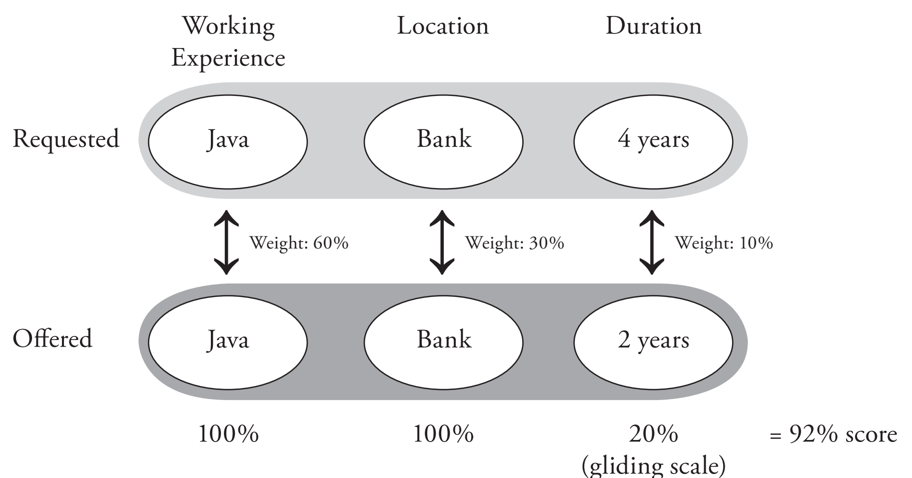
Please refer to the *"StringSetMatchRule"* and the *"MatrixMatchRule"* documentation for more information.

IntersectMatchRule

The match rules described so far allow you to match for example on skills and years of experience. However, it is not possible to match combinations of attributes. For example, you could match employees with Java experience and employees with 5 years of working experience, but it is not possible to match employees with "5 years of Java experience" or "2 years of C++ experience".

The IntersectMatchRule makes this possible.

The example below shows a match based on working experience, working location and working duration. In this example, we look for an employee with 4 years of Java experience at a bank. We match this request with an employee with 2 years of experience at a bank.



Each part of the match can be assigned a weight that indicates the importance of each part. Please don't confuse this with the weight of the match rule itself that indicates the importance of the Intersect match rule within the complete match.

The match engine processes each part from left to right. When the first part results in a 0% score, the remaining parts will be skipped and the matchrule results in a 0% score.

Match rules

The intersect match rule is implemented in the class *"com.match4j.matchrules.IntersectMatchRule"*.

To use this match rule, it has to be configured in the Match4J config file:

```
<matchrule>
    ...
    <implementingclass>
        <classname>
            com.match4j.matchrules.IntersectMatchRule
        </classname>
        <parameter>
                <key>RequestAttributeId</key>
                <value>workexperiences</value>
        </parameter>
        <parameter>
            <key>ResultAttributeId</key>
            <value>workexperiences</value>
        </parameter>
        <parameter>
            <key>SetDelimiter</key>
            <value>,</value>
        </parameter>
        <parameter>
            <key>IntersectDelimiter</key>
            <value>;</value>
        </parameter>
        <parameter>
            <key>IntersectConfig</key>
            <value>intersect.conf</value>
        </parameter>
    </implementingclass>
</matchrule>
```

There are two several parameters: The *"RequestAttributeId"* and *"ResultAttributeId"* define the attributes of the request entity and of the result entity, respectively. Only String type attributes are allowed.

Next, you have to define two delimiters. The *"IntersectDelimiter"* defines the delimer to separate the parts like *"experience"*, *"location"* and "duration". An entity can represent several of these combinations. These will be separated by the delimiter you specified in the *"SetDelimiter"* parameter.

Finally the *"IntersectConfig"* parameter defines the file path for the match rule configuration details. The intersect configuration file is an XML document that should comply to the DTD as defined in *"document/config/intersect-config.dtd"*.

Match rules

The example below shows the definition of a intersect match rule that can be used to match the working experience.

```
<intersect>
        <level>
                <!-- skill -->
                <weight>60</weight>
                <matchtype>SIMPLE</matchtype>
        </level>

        <level>
                <!-- location -->
                <weight>30</weight>
                <matchtype>MATRIX</matchtype>
                <parameter>
                        <key>MatrixConfig</key>
                        <value>matrix.conf</value>
                </parameter>
        </level>

        <level>
                <!-- duration -->
                <weight>10</weight>
                <matchtype>SIMPLE</matchtype>
        </level>
</intersect>
```

Writing your own match rules

It is possible to write custom match rules. For example, this makes it possible to use a zipcode table to match on travel distance.

Complete the following steps in order to implement a custom match rule:

1. create a new class, for example, *"MyMatchRule"* and have this class implement the *"com.match4j.IMatchRule"* interface. See the JavaDoc of this interface for a description of the methods.

2. compile the class and incorporate it in the classpath of the application in which the match rule should be used. Also incorporate the fully qualified class name in the configuration file of that same application.