

Data caches - Introduction and Configuration

The architecture of Match4J is designed in such a way that the match engine does not require information on the origin of the data it performs the matching on.

In order to attain this, the system utilizes so-called “data caches”.

For every data source, a data cache can be implemented, which enables the match engine to retrieve the data appropriately.

A data cache implementation, which can retrieve data from relational databases by means of JDBC, is supplied in the package. In addition, the supplied example application “*simple*” contains a very simple implementation of a data cache.

When you want to retrieve data from other data sources (for instance, from a FTP server or mainframe database), you can implement your own data cache (or have it designed by Match4J specialists).

Configuring data caches

The data cache which is to be used should be indicated in the configuration file of Match4J. More information on this subject is provided in the configuration manual.

Database Replication Cache

Match4J is supplied with a data cache implementation, that can retrieve data from a relational database, for which a JDBC driver is available. The functional specification of this data cache are:

- by means of an XML configuration file, a mapping between table columns and entity types can be defined.
- the cache can be periodically refreshed.
- this data cache retrieves data from only one table. When retrieving data from more than one table views should be implemented in the database.

To use this data cache, you have to configure the following in the Match4J configuration file:

```
<cache>
  <classname>
    com.match4j.data.cache.DatabaseReplicationCache
  </classname>
  <parameter>
    <key>config</key>
    <value>config/databaseReplicationCache.conf</value>
  </parameter>
</cache>
```

This data cache implementation expects only one parameter, the filepath to a config file that defines the data cache configuration. For more information please consult the configuration manual (match4j_config.pdf).

The configuration file is an XML document that should comply with the DTD

Data caches - Introduction and Configuration

as defined in “*documents/config/databaseReplicationCache-config.dtd*” and has the following structure:

```
<matchengine>
  <datacache>
    <refreshintervalminutes>...</refreshintervalminutes>
    <datasource>...</datasource>
    <datamappings>...</datamappings>
  </datacache>
</matchengine>
```

The “*refreshintervalminutes*” tag defines how often the data should be reloaded from the database.

The “*datasource*” tag defines which JDBC database to use. Please consult JDBC documentation for more information.

```
<datasource>
  <databasedriver>...</databasedriver>
  <databaseurl>...</databaseurl>
  <username>...</username>
  <password>...</password>
</datasource>
```

Finally, the “*datamappings*” tag defines the mapping between tables from the database and entity types.

```
<datamappings>
  <datamapping> (1 or more)
    <entityid>...</entityid>
    <tablename>...</tablename>
    <keycolumn>...</keycolumn>
    <mapping...</mapping> (1 or more)
  </datamapping>
</datamappings>
```

For each mapping between a table and an entity type, a “*datamapping*” tag is used. This element consists of an “*entityid*” and a “*tablename*” tag which represents the id of the entity type and which table to use. The “*keycolumn*” defines which column of the table contains unique values. Those values will be used as the unique id’s of the entities. **Note:** a table is required to have a key column when used by this data cache.

Finally, 1 or more “*mapping*” tags are defined, that map a column of the database table to an attribute of the entity type.

Data caches - Introduction and Configuration

```
<mapping>
  <attributeid>...</attributeid>
  <columnname>...</columnname>
</mapping>
```

The example below defines a mapping between a “CAR” table and a “Car” entity type. The “REGISTRATION_NUMBER” column of the table will be mapped to the “id” of the entity type. Furthermore, it defines a mapping between the brand and sunroof columns/attributes.

This example also defines that the cache will be updated every minute.

```
<matchengine>
  <datacache>
    <refreshintervalminutes>1</refreshintervalminutes>

    <datasource>
      <databasedriver>
        org.hsqldb.jdbcDriver</databasedriver>
      <databaseurl>jdbc:hsqldb:persons</databaseurl>
      <username>a_username</username>
      <password>a_password</password>
    </datasource>

    <datamappings>
      <datamapping>
        <entityid>Car</entityid>
        <tabelname>CARS</tabelname>
        <keycolumn>REGISTRATION_NUMBER</keycolumn>
        <mapping>
          <attributeid>brand</attributeid>
          <columnname>BRAND</columnname>
        </mapping>
        <mapping>
          <attributeid>sunroof</attributeid>
          <columnname>SUNROOF</columnname>
        </mapping>
      </datamapping>
    </datamappings>
  </datacache>
</matchengine>
```

Data caches - Introduction and Configuration

Writing your own data cache

It takes little effort to design a custom data cache. For an example, look at the “*SimpleExampleDataCache*” class in the “*examples/simple*” directory.

Complete the following steps in order to implement a new data cache:

1. create a new class, for example, “*MyDataCache*” and have this class implement the “*com.match4j.IDataCache*” interface.
2. implement the method “*void setProperties(Properties aProperties)*”. This method may be empty, but it allows for reading the configuration.
3. implement the “*Iterator getDataIterator(String aEntityId, Properties aProperties)*” method. This should return an iteration of entity objects for the requested entity type.
4. implement the “*void setEntityTypes(Hashtable entityTypes)*” method. This method, called by the match engine only once, passes the entity type objects.
5. compile the class and incorporate it in the classpath of the application in which the data cache should be used. Also incorporate the fully qualified class name in the configuration file of that same application.

Important

- the “*getData(...)*” method is called for each match request. Therefore, it is highly recommended to register the data objects in the cache, not only because of performance reasons, but also because of memory use.
- the “*getData(...)*” method should be implemented as being thread safe, because the match engine can simultaneously call upon this method from multiple threads.