MyBatis Example

MyBatis Example

Available as of Camel 2.12

This example is located in the `examples/camel-example-mybatis` directory of the Camel distribution.
There is a `README.txt` file with instructions how to run it.

If you use maven then you can easily install the example from the command line:

```
mvn install
```

About

This example shows how to exchange data using a shared database table.

The example has two Camel routes. The first route insert new data into the table, triggered by a timer to run every 5th second.
The second route pickup the newly inserted rows from the table, process the row(s), and mark the row(s) as processed when done; to avoid picking up the same rows again.

Implementation

In the `camel-mybatis.xml` file in the `src/main/resources/OSGI-INF/blueprint` folder we have the Blueprint XML file. This example uses an embedded Database so we use the following bean to create and drop the table(s).

```
 <bean id="initDatabase" class="org.apache.camel.example.mybatis.DatabaseBean"
        init-method="create" destroy-method="destroy">
    <property name="camelContext" ref="myBatisAndCamel"/>
  </bean>
```

This example uses a bean to generate orders

```
 <bean id="orderService" class="org.apache.camel.example.mybatis.OrderService"/>
```

And the CamelContext has two routes as shown below:

```
  <camelContext id="myBatisAndCamel" xmlns="http://camel.apache.org/schema/blueprint">

    <!-- route that generate new orders and insert them in the database -->
    <route id="generateOrder-route">
      <from uri="timer:foo?period=5s"/>
      <transform>
        <method ref="orderService" method="generateOrder"/>
      </transform>
      <to uri="mybatis:insertOrder?statementType=Insert"/>
      <log message="Inserted new order ${body.id}"/>
    </route>

    <!-- route that process the orders by picking up new rows from the database
         and when done processing then update the row to mark it as processed -->
    <route id="processOrder-route">
      <from uri="mybatis:selectOrders?statementType=SelectList&amp;consumer.onConsume=consumeOrder"/>
      <to uri="bean:orderService?method=processOrder"/>
      <log message="${body}"/>
    </route>

  </camelContext>
```

MyBatis SqlMapConfig.xml

MyBatis is configured using a `SqlMapConfig.xml` file located in the root of the classpath, eg in `src/main/resources`.
This configuration files setup MyBatis as well a pooled data source

MyBatis SqlMapConfig.xml

```
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

  <settings>
    <setting name="useGeneratedKeys" value="false"/>
  </settings>

  <!-- Use type aliases to avoid typing the full classname every time. -->
  <typeAliases>
    <typeAlias alias="Order" type="org.apache.camel.example.mybatis.Order"/>
  </typeAliases>

  <!-- setup environment with JDBC data source -->
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
        <property name="url" value="jdbc:derby:memory:mybatis;create=true"/>
      </dataSource>
    </environment>
  </environments>

  <!-- mapping files -->
  <mappers>
    <mapper resource="org/apache/camel/example/mybatis/Order.xml"/>
  </mappers>

</configuration>
```

MyBatis mapping files

MyBatis allows to externalize the SQL queries and mapping from SQL to POJOs.

We have a plain POJO `org.apache.camel.example.mybatis.Order` which just has getter/setters as shown below:

Order POJO

```
public class Order {

    private int id;
    private String item;
    private int amount;
    private String description;
    private boolean processed;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getItem() {
        return item;
    }

    public void setItem(String item) {
        this.item = item;
    }

    public int getAmount() {
        return amount;
    }

    public void setAmount(int amount) {
        this.amount = amount;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public boolean isProcessed() {
        return processed;
    }

    public void setProcessed(boolean processed) {
        this.processed = processed;
    }
}
```

And the MyBatis mapping file `Order.xml` is located in `src/main/resources/org/apache/camel/example/mybatis` where we map from SQL to this Order POJO, as shown below:

MyBatis mapping file for Order

```
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="Order">

  <!-- Result maps describe the mapping between the columns returned
 from a query, and the class properties.  A result map isn't
 necessary if the columns (or aliases) match to the properties
 exactly. -->
  <resultMap id="OrderResult" type="Order">
    <result property="id" column="ORD_ID"/>
    <result property="item" column="ITEM"/>
    <result property="amount" column="ITEM_COUNT"/>
    <result property="description" column="ITEM_DESC"/>
    <result property="processed" column="ORD_DELETED"/>
  </resultMap>

  <!-- Select with no parameters using the result map for Order class. -->
  <select id="selectOrders" resultMap="OrderResult">
    select * from ORDERS where ORD_DELETED = false order by ORD_ID
  </select>

  <!-- Insert example, using the Order parameter class -->
  <insert id="insertOrder" parameterType="Order">
    insert into ORDERS (
    ORD_ID,
    ITEM,
    ITEM_COUNT,
    ITEM_DESC,
    ORD_DELETED
    )
    values (
    #{id}, #{item}, #{amount}, #{description}, false
    )
  </insert>

  <update id="consumeOrder" parameterType="Order">
    update ORDERS set ORD_DELETED = true where ORD_ID = #{id}
  </update>

</mapper>
```

Running the example

This example requires running in Apache Karaf / ServiceMix

To install Apache Camel in Karaf you type in the shell (we use version 2.12.0):

```
  features:chooseurl camel 2.12.0
  features:install camel
```

First you need to install the following features in Karaf/ServiceMix with:

```
  features:install camel-mybatis
```

Then you can install the Camel example:

```
  osgi:install -s mvn:org.apache.camel/camel-example-mybatis/2.12.0
```

And you can see the application running by tailing the logs

```
  log:tail
```

And you can use ctrl + c to stop tailing the log.

As of Camel 2.12.3 onwards

You can install and run this example using it's `features.xml` with the following 2 shell commands (substitute the `${version}` placeholder below with the concrete version of Camel in use):

```
features:addUrl mvn:org.apache.camel/camel-example-mybatis/${version}/xml/features
features:install camel-example-mybatis
```

See Also

- Examples
- MyBatis
- SQL Example
- Hibernate Example