MyBatis

MyBatis

Available as of Camel 2.7

The **mybatis:** component allows you to query, poll, insert, update and delete data in a relational database using MyBatis.

Maven users will need to add the following dependency to their `pom.xml` for this component:

xml<dependency> <groupId>org.apache.camel</groupId> <artifactId>camel-mybatis</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency>

URI format

mybatis:statementName[?options]

Where **statementName** is the statement name in the MyBatis XML mapping file which maps to the query, insert, update or delete operation you wish to evaluate.

You can append query options to the URI in the following format, `?option=value&option=value&...`

This component will by default load the MyBatis SqlMapConfig file from the root of the classpath with the expected name of `SqlMapConfig.xml`.
If the file is located in another location, you will need to configure the `configurationUri` option on the `MyBatisComponent` component.

Options

confluenceTableSmall

Option	Type	Default	Description
`consumer.onConsume`	`String`	`null`	Statements to run after consuming. Can be used, for example, to update rows after they have been consumed and processed in Camel. See sample later. Multiple statements can be separated with commas.
`consumer.useIterator`	`boolean`	`true`	If `true` each row returned when polling will be processed individually. If `false` the entire `List` of data is set as the IN body.
`consumer.routeEmptyResultSet`	`boolean`	`false`	Sets whether empty result sets should be routed.
`statementType`	`StatementType`	`null`	Mandatory to specify for the producer to control which kind of operation to invoke. The enum values are: `SelectOne`, `SelectList`, `Insert`, `InsertList`, `Update`, `UpdateList`, `Delete`, and `DeleteList`. **Notice:** `InsertList` is available as of Camel 2.10, and `UpdateList`, `DeleteList` is available as of Camel 2.11.
`maxMessagesPerPoll`	`int`	`0`	This option is intended to split results returned by the database pool into the batches and deliver them in multiple exchanges. This integer defines the maximum messages to deliver in single exchange. By default, no maximum is set. Can be used to set a limit of e.g. 1000 to avoid when starting up the server that there are thousands of files. Set a value of 0 or negative to disable it.
`executorType`	`String`	`null`	**Camel 2.11:** The executor type to be used while executing statements. The supported values are: simple, reuse, batch. By default, the value is not specified and is equal to what MyBatis uses, i.e. **simple**. **simple** executor does nothing special. **reuse** executor reuses prepared statements. **batch** executor reuses statements and batches updates.
`outputHeader`	`String`	`null`	**Camel 2.15:** To store the result as a header instead of the message body. This allows to preserve the existing message body as-is.
`inputHeader`	`String`	`null`	**Camel 2.15:** "inputHeader" parameter to use a header value as input to the component instead of the body.
`transacted`	`boolean`	`false`	**Camel 2.16.2: SQL consumer only:**Enables or disables transaction. If enabled then if processing an exchange failed then the consumer break out processing any further exchanges to cause a rollback eager

Message Headers

Camel will populate the result message, either IN or OUT with a header with the statement used:

confluenceTableSmall

Header	Type	Description
`CamelMyBatisStatementName`	`String`	The **statementName** used (for example: insertAccount).
`CamelMyBatisResult`	`Object`	The **response** returned from MtBatis in any of the operations. For instance an `INSERT` could return the auto-generated key, or number of rows etc.

Message Body

The response from MyBatis will only be set as the body if it's a `SELECT` statement. That means, for example, for `INSERT` statements Camel will not replace the body. This allows you to continue routing and keep the original body. The response from MyBatis is always stored in the header with the key `CamelMyBatisResult`.

Samples

For example if you wish to consume beans from a JMS queue and insert them into a database you could do the following:

from("activemq:queue:newAccount"). to("mybatis:insertAccount?statementType=Insert");

Notice we have to specify the `statementType`, as we need to instruct Camel which kind of operation to invoke.

Where **insertAccount** is the MyBatis ID in the SQL mapping file:

xml <!-- Insert example, using the Account parameter class --> <insert id="insertAccount" parameterType="Account"> insert into ACCOUNT (ACC_ID, ACC_FIRST_NAME, ACC_LAST_NAME, ACC_EMAIL) values (#{id}, #{firstName}, #{lastName}, #{emailAddress}) </insert>

Using StatementType for better control of MyBatis

When routing to an MyBatis endpoint you will want more fine grained control so you can control whether the SQL statement to be executed is a `SELECT`, `UPDATE`, `DELETE` or `INSERT` etc. So for instance if we want to route to an MyBatis endpoint in which the IN body contains parameters to a `SELECT` statement we can do:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mybatis/src/test/java/org/apache/camel/component/mybatis/MyBatisSelectOneTest.java}In the code above we can invoke the MyBatis statement `selectAccountById` and the IN body should contain the account id we want to retrieve, such as an `Integer` type.

We can do the same for some of the other operations, such as `SelectList`:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mybatis/src/test/java/org/apache/camel/component/mybatis/MyBatisSelectListTest.java}And the same for `UPDATE`, where we can send an `Account` object as the IN body to MyBatis:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mybatis/src/test/java/org/apache/camel/component/mybatis/MyBatisUpdateTest.java}

Using InsertList StatementType

Available as of Camel 2.10

MyBatis allows you to insert multiple rows using its for-each batch driver. To use this, you need to use the <foreach> in the mapper XML file. For example as shown below:{snippet:id=insertList|lang=xml|url=camel/trunk/components/camel-mybatis/src/test/resources/org/apache/camel/component/mybatis/Account.xml}Then you can insert multiple rows, by sending a Camel message to the `mybatis` endpoint which uses the `InsertList` statement type, as shown below:{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mybatis/src/test/java/org/apache/camel/component/mybatis/MyBatisInsertListTest.java}

Using UpdateList StatementType

Available as of Camel 2.11

MyBatis allows you to update multiple rows using its for-each batch driver. To use this, you need to use the <foreach> in the mapper XML file. For example as shown below:

xml<update id="batchUpdateAccount" parameterType="java.util.Map"> update ACCOUNT set ACC_EMAIL = #{emailAddress} where ACC_ID in <foreach item="Account" collection="list" open="(" close=")" separator=","> #{Account.id} </foreach> </update>

Then you can update multiple rows, by sending a Camel message to the mybatis endpoint which uses the UpdateList statement type, as shown below:

from("direct:start") .to("mybatis:batchUpdateAccount?statementType=UpdateList") .to("mock:result");

Using DeleteList StatementType

Available as of Camel 2.11

MyBatis allows you to delete multiple rows using its for-each batch driver. To use this, you need to use the <foreach> in the mapper XML file. For example as shown below:

xml<delete id="batchDeleteAccountById" parameterType="java.util.List"> delete from ACCOUNT where ACC_ID in <foreach item="AccountID" collection="list" open="(" close=")" separator=","> #{AccountID} </foreach> </delete>

Then you can delete multiple rows, by sending a Camel message to the mybatis endpoint which uses the DeleteList statement type, as shown below:

from("direct:start") .to("mybatis:batchDeleteAccount?statementType=DeleteList") .to("mock:result");

Notice on InsertList, UpdateList and DeleteList StatementTypes

Parameter of any type (List, Map, etc.) can be passed to mybatis and an end user is responsible for handling it as required with the help of mybatis dynamic queries capabilities.

Scheduled polling example

This component supports scheduled polling and can therefore be used as a Polling Consumer. For example to poll the database every minute:

from("mybatis:selectAllAccounts?delay=60000").to("activemq:queue:allAccounts");

See "ScheduledPollConsumer Options" on Polling Consumer for more options.

Alternatively you can use another mechanism for triggering the scheduled polls, such as the Timer or Quartz components. In the sample below we poll the database, every 30 seconds using the Timer component and send the data to the JMS queue:

javafrom("timer://pollTheDatabase?delay=30000").to("mybatis:selectAllAccounts").to("activemq:queue:allAccounts");

And the MyBatis SQL mapping file used:

xml <!-- Select with no parameters using the result map for Account class. --> <select id="selectAllAccounts" resultMap="AccountResult"> select * from ACCOUNT </select>

Using onConsume

This component supports executing statements **after** data have been consumed and processed by Camel. This allows you to do post updates in the database. Notice all statements must be `UPDATE` statements. Camel supports executing multiple statements whose names should be separated by commas.

The route below illustrates we execute the **consumeAccount** statement data is processed. This allows us to change the status of the row in the database to processed, so we avoid consuming it twice or more.{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mybatis/src/test/java/org/apache/camel/component/mybatis/MyBatisQueueTest.java}And the statements in the sqlmap file:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-mybatis/src/test/resources/org/apache/camel/component/mybatis/Account.xml}{snippet:id=e2|lang=xml|url=camel/trunk/components/camel-mybatis/src/test/resources/org/apache/camel/component/mybatis/Account.xml}

Participating in transactions

Setting up a transaction manager under camel-mybatis can be a little bit fiddly, as it involves externalising the database configuration outside the standard MyBatis `SqlMapConfig.xml` file.

The first part requires the setup of a `DataSource`. This is typically a pool (either DBCP, or c3p0), which needs to be wrapped in a Spring proxy. This proxy enables non-Spring use of the `DataSource` to participate in Spring transactions (the MyBatis `SqlSessionFactory` does just this).

xml <bean id="dataSource" class="org.springframework.jdbc.datasource.TransactionAwareDataSourceProxy"> <constructor-arg> <bean class="com.mchange.v2.c3p0.ComboPooledDataSource"> <property name="driverClass" value="org.postgresql.Driver"/> <property name="jdbcUrl" value="jdbc:postgresql://localhost:5432/myDatabase"/> <property name="user" value="myUser"/> <property name="password" value="myPassword"/> </bean> </constructor-arg> </bean>

This has the additional benefit of enabling the database configuration to be externalised using property placeholders.

A transaction manager is then configured to manage the outermost `DataSource`:

xml <bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="dataSource"/> </bean>

A mybatis-spring `SqlSessionFactoryBean` then wraps that same `DataSource`:

xml <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"> <property name="dataSource" ref="dataSource"/> <!-- standard mybatis config file --> <property name="configLocation" value="/META-INF/SqlMapConfig.xml"/> <!-- externalised mappers --> <property name="mapperLocations" value="classpath*:META-INF/mappers/**/*.xml"/> </bean>

The camel-mybatis component is then configured with that factory:

xml <bean id="mybatis" class="org.apache.camel.component.mybatis.MyBatisComponent"> <property name="sqlSessionFactory" ref="sqlSessionFactory" /> </bean>

Finally, a transaction policy is defined over the top of the transaction manager, which can then be used as usual:

xml <bean id="PROPAGATION_REQUIRED" class="org.apache.camel.spring.spi.SpringTransactionPolicy"> <property name="transactionManager" ref="txManager"/> <property name="propagationBehaviorName" value="PROPAGATION_REQUIRED"/> </bean> <camelContext id="my-model-context" xmlns="http://camel.apache.org/schema/spring"> <route id="insertModel"> <from uri="direct:insert"/> <transacted ref="PROPAGATION_REQUIRED"/> <to uri="mybatis:myModel.insert?statementType=Insert"/> </route> </camelContext>

Endpoint See Also