MINA

MINA Component

Deprecated

Deprecated
This component is deprecated as the Apache Mina 1.x project is EOL. Instead use MINA2 or Netty instead.

The **mina:** component is a transport for working with Apache MINA

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-mina</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency>

URI format

mina:tcp://hostname[:port][?options] mina:udp://hostname[:port][?options] mina:vm://hostname[:port][?options]

You can specify a codec in the Registry using the **codec** option. If you are using TCP and no codec is specified then the `textline` flag is used to determine if text line based codec or object serialization should be used instead. By default the object serialization is used.

For UDP if no codec is specified the default uses a basic `ByteBuffer` based codec.

The VM protocol is used as a direct forwarding mechanism in the same JVM. See the MINA VM-Pipe API documentation for details.

A Mina producer has a default timeout value of 30 seconds, while it waits for a response from the remote server.

In normal use, `camel-mina` only supports marshalling the body content—message headers and exchange properties are not sent.
However, the option, **transferExchange**, does allow you to transfer the exchange itself over the wire. See options below.

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

Options

confluenceTableSmall

Option	Default Value	Description
`codec`	`null`	You can refer to a named `ProtocolCodecFactory` instance in your Registry such as your Spring `ApplicationContext`, which is then used for the marshalling.
`codec`	`null`	You must use the # notation to look up your codec in the Registry. For example, use `#myCodec` to look up a bean with the `id` value, `myCodec`.
`disconnect`	`false`	**Camel 2.3:** Whether or not to disconnect(close) from Mina session right after use. Can be used for both consumer and producer.
`textline`	`false`	Only used for TCP. If no codec is specified, you can use this flag to indicate a text line based codec; if not specified or the value is `false`, then Object Serialization is assumed over TCP.
`textlineDelimiter`	`DEFAULT`	Only used for TCP and if **textline=true**. Sets the text line delimiter to use. Possible values are: `DEFAULT`, `AUTO`, `WINDOWS`, `UNIX` or `MAC`. If none provided, Camel will use `DEFAULT`. This delimiter is used to mark the end of text.
`sync`	`true`	Setting to set endpoint as one-way or request-response.
`lazySessionCreation`	`true`	Sessions can be lazily created to avoid exceptions, if the remote server is not up and running when the Camel producer is started.
`timeout`	`30000`	You can configure the timeout that specifies how long to wait for a response from a remote server. The timeout unit is in milliseconds, so 60000 is 60 seconds. The timeout is only used for Mina producer.
`encoding`	*JVM Default*	You can configure the encoding (a charset name) to use for the TCP textline codec and the UDP protocol. If not provided, Camel will use the JVM default Charset.
`transferExchange`	`false`	Only used for TCP. You can transfer the exchange over the wire instead of just the body. The following fields are transferred: In body, Out body, fault body, In headers, Out headers, fault headers, exchange properties, exchange exception. This requires that the objects are *serializable*. Camel will exclude any non-serializable objects and log it at `WARN` level.
`minaLogger`	`false`	You can enable the Apache MINA logging filter. Apache MINA uses `slf4j` logging at `INFO` level to log all input and output.

`filters`	`null`	You can set a list of Mina IoFilters to register. The `filters` value must be one of the following: • **Camel 2.2:** comma-separated list of bean references (e.g. `#filterBean1,#filterBean2`) where each bean must be of type `org.apache.mina.common.IoFilter`. • **before Camel 2.2:** a reference to a bean of type `List<org.apache.mina.common.IoFilter>`.
`encoderMaxLineLength`	`-1`	As of 2.1, you can set the textline protocol encoder max line length. By default the default value of Mina itself is used which are `Integer.MAX_VALUE`.
`decoderMaxLineLength`	`-1`	As of 2.1, you can set the textline protocol decoder max line length. By default the default value of Mina itself is used which are 1024.
`producerPoolSize`	16	The TCP producer is thread safe and supports concurrency much better. This option allows you to configure the number of threads in its thread pool for concurrent producers. **Note:** Camel has a pooled service which ensured it was already thread safe and supported concurrency already.
`allowDefaultCodec`	`true`	The mina component installs a default codec if both, `codec` is `null` and `textline` is `false`. Setting `allowDefaultCodec` to `false` prevents the mina component from installing a default codec as the first element in the filter chain. This is useful in scenarios where another filter must be the first in the filter chain, like the SSL filter.
`disconnectOnNoReply`	`true`	**Camel 2.3:** If sync is enabled then this option dictates MinaConsumer if it should disconnect where there is no reply to send back.
`noReplyLogLevel`	`WARN`	**Camel 2.3:** If sync is enabled this option dictates MinaConsumer which logging level to use when logging a there is no reply to send back. Values are: `FATAL, ERROR, INFO, DEBUG, OFF`.
`clientMode`	`false`	**Camel 2.15:** Consumer only. If the `clientMode` is true, mina consumer will connect the address as a TCP client.

Using a custom codec

See the Mina documentation how to write your own codec. To use your custom codec with `camel-mina`, you should register your codec in the Registry; for example, by creating a bean in the Spring XML file. Then use the `codec` option to specify the bean ID of your codec. See HL7 that has a custom codec.

Sample with sync=false

In this sample, Camel exposes a service that listens for TCP connections on port 6200. We use the **textline** codec. In our route, we create a Mina consumer endpoint that listens on port 6200:

{snippet:id=e1|lang=java|url=camel/trunk/components/camel-mina/src/test/java/org/apache/camel/component/mina/MinaConsumerTest.java}

As the sample is part of a unit test, we test it by sending some data to it on port 6200.

{snippet:id=e2|lang=java|url=camel/trunk/components/camel-mina/src/test/java/org/apache/camel/component/mina/MinaConsumerTest.java}

Sample with sync=true

In the next sample, we have a more common use case where we expose a TCP service on port 6201 also use the textline codec. However, this time we want to return a response, so we set the `sync` option to `true` on the consumer.

{snippet:id=e3|lang=java|url=camel/trunk/components/camel-mina/src/test/java/org/apache/camel/component/mina/MinaConsumerTest.java}

Then we test the sample by sending some data and retrieving the response using the `template.requestBody()` method. As we know the response is a `String`, we cast it to `String` and can assert that the response is, in fact, something we have dynamically set in our processor code logic.

{snippet:id=e4|lang=java|url=camel/trunk/components/camel-mina/src/test/java/org/apache/camel/component/mina/MinaConsumerTest.java}

Sample with Spring DSL

Spring DSL can, of course, also be used for MINA. In the sample below we expose a TCP server on port 5555:

xml <route> <from uri="mina:tcp://localhost:5555?textline=true"/> <to uri="bean:myTCPOrderHandler"/> </route>

In the route above, we expose a TCP server on port 5555 using the textline codec. We let the Spring bean with ID, `myTCPOrderHandler`, handle the request and return a reply. For instance, the handler bean could be implemented as follows:

java public String handleOrder(String payload) { ... return "Order: OK" }

Configuring Mina endpoints using Spring bean style

Configuration of Mina endpoints is possible using regular Spring bean style configuration in the Spring DSL.

However, in the underlying Apache Mina toolkit, it is relatively difficult to set up the acceptor and the connector, because you can *not* use simple setters. To resolve this difficulty, we leverage the `MinaComponent` as a Spring factory bean to configure this for us. If you really need to configure this yourself, there are setters on the `MinaEndpoint` to set these when needed.

The sample below shows the factory approach:

{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-mina/src/test/resources/org/apache/camel/component/mina/SpringMinaEndpointTest-context.xml}

And then we can refer to our endpoint directly in the route, as follows:

{snippet:id=e2|lang=xml|url=camel/trunk/components/camel-mina/src/test/resources/org/apache/camel/component/mina/SpringMinaEndpointTest-context.xml}

Closing Session When Complete

When acting as a server you sometimes want to close the session when, for example, a client conversion is finished. To instruct Camel to close the session, you should add a header with the key `CamelMinaCloseSessionWhenComplete` set to a boolean `true` value.

For instance, the example below will close the session after it has written the `bye` message back to the client:

java from("mina:tcp://localhost:8080?sync=true&textline=true").process(new Processor() { public void process(Exchange exchange) throws Exception { String body = exchange.getIn().getBody(String.class); exchange.getOut().setBody("Bye " + body); exchange.getOut().setHeader(MinaConstants.MINA_CLOSE_SESSION_WHEN_COMPLETE, true); } });

Get the IoSession for message

Available since Camel 2.1
You can get the IoSession from the message header with this key MinaEndpoint.HEADER_MINA_IOSESSION, and also get the local host address with the key MinaEndpoint.HEADER_LOCAL_ADDRESS and remote host address with the key MinaEndpoint.HEADER_REMOTE_ADDRESS.

Configuring Mina filters

Filters permit you to use some Mina Filters, such as `SslFilter`. You can also implement some customized filters. Please note that `codec` and `logger` are also implemented as Mina filters of type, `IoFilter`. Any filters you may define are appended to the end of the filter chain; that is, after `codec` and `logger`.

If using the `SslFilter` you need to add the `mina-filter-ssl` JAR to the classpath.

For instance, the example below will send a keep-alive message after 10 seconds of inactivity:

javapublic class KeepAliveFilter extends IoFilterAdapter { @Override public void sessionCreated(NextFilter nextFilter, IoSession session) throws Exception { session.setIdleTime(IdleStatus.BOTH_IDLE, 10); nextFilter.sessionCreated(session); } @Override public void sessionIdle(NextFilter nextFilter, IoSession session, IdleStatus status) throws Exception { session.write("NOOP"); // NOOP is a FTP command for keep alive nextFilter.sessionIdle(session, status); } }

As Camel Mina may use a request-reply scheme, the endpoint as a client would like to drop some message, such as greeting when the connection is established. For example, when you connect to an FTP server, you will get a `220` message with a greeting (`220 Welcome to Pure-FTPd`). If you don't drop the message, your request-reply scheme will be broken.

javapublic class DropGreetingFilter extends IoFilterAdapter { @Override public void messageReceived(NextFilter nextFilter, IoSession session, Object message) throws Exception { if (message instanceof String) { String ftpMessage = (String) message; // "220" is given as greeting. "200 Zzz" is given as a response to "NOOP" (keep alive) if (ftpMessage.startsWith("220") || or ftpMessage.startsWith("200 Zzz")) { // Dropping greeting return; } } nextFilter.messageReceived(session, message); } }

Then, you can configure your endpoint using Spring DSL:

xml<bean id="myMinaFactory" class="org.apache.camel.component.mina.MinaComponent"> <constructor-arg index="0" ref="camelContext" /> </bean> <bean id="myMinaEndpoint" factory-bean="myMinaFactory" factory-method="createEndpoint"> <constructor-arg index="0" ref="myMinaConfig"/> </bean> <bean id="myMinaConfig" class="org.apache.camel.component.mina.MinaConfiguration"> <property name="protocol" value="tcp" /> <property name="host" value="localhost" /> <property name="port" value="2121" /> <property name="sync" value="true" /> <property name="minaLogger" value="true" /> <property name="filters" ref="listFilters"/> </bean> <bean id="listFilters" class="java.util.ArrayList" > <constructor-arg> <list value-type="org.apache.mina.common.IoFilter"> <bean class="com.example.KeepAliveFilter"/> <bean class="com.example.DropGreetingFilter"/> </list> </constructor-arg> </bean>

Endpoint See Also

- MINA2
- Netty