# Idempotent Consumer

## Idempotent Consumer

The Idempotent Consumer from the EIP patterns is used to filter out duplicate messages.

This pattern is implemented using the IdempotentConsumer class. This uses an Expression to calculate a unique message ID string for a given message exchange; this ID can then be looked up in the IdempotentRepository to see if it has been seen before; if it has the message is consumed; if its not then the message is processed and the ID is added to the repository.

The Idempotent Consumer essentially acts like a Message Filter to filter out duplicates.

Camel will add the message id eagerly to the repository to detect duplication also for Exchanges currently in progress.
On completion Camel will remove the message id from the repository if the Exchange failed, otherwise it stays there.

Camel provides the following Idempotent Consumer implementations:

- MemoryIdempotentRepository
- FileIdempotentRepository
- HazelcastIdempotentRepository (**Available as of Camel 2.8**)
- JdbcMessageIdRepository (**Available as of Camel 2.7**)
- JpaMessageIdRepository
- InfinispanIdempotentRepository (**Available as of Camel 2.13.0)**
- JCacheIdempotentRepository **(Available as of Camel 2.17.0)**
- SpringCacheIdempotentRepository **(Available as of Camel 2.17.1)**
- EhcacheIdempotentRepository **(Available as of Camel 2.18.0)**
- KafkaIdempotentRepository (**Available as of Camel 2.19.0)**

## Options

The Idempotent Consumer has the following options:

| Option | Default | Description |
|---|---|---|
| eager | true | Eager controls whether Camel adds the message to the repository before or after the exchange has been processed. If enabled before then Camel will be able to detect duplicate messages even when messages are currently in progress. By disabling Camel will only detect duplicates when a message has successfully been processed. |
| messageIdRepositoryRef | `null` | A reference to a `IdempotentRepository` to lookup in the registry. This option is mandatory when using XML DSL. |
| skipDuplicate | true | **Camel 2.8:** Sets whether to skip duplicate messages. If set to `false` then the message will be continued. However the Exchange has been marked as a duplicate by having the `Exchange.DUPLICATE_MESSAG` exchange property set to a `Boolean.TRUE` value. |
| removeOnFailure | true | **Camel 2.9:** Sets whether to remove the id of an Exchange that failed. |
| completionEager | false | **Camel 2.16:** Sets whether to complete the idempotent consumer eager or when the exchange is done.<br><br>If this option is true to complete eager, then the idempotent consumer will trigger its completion when the exchange reached the end of the block of the idempotent consumer pattern. So if the exchange is continued routed after the block ends, then whatever happens there does not affect the state.<br><br>If this option is false (default) to not complete eager, then the idempotent consumer will complete when the exchange is done being routed. So if the exchange is continued routed after the block ends, then whatever happens there also affect the state. For example if the exchange failed due to an exception, then the state of the idempotent consumer will be a rollback. |

## Using the Fluent Builders

The following example will use the header **myMessageId** to filter out duplicates{snippet:id=idempotent|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/RouteBuilderTest.java}The above example will use an in-memory based MessageIdRepository which can easily run out of memory and doesn't work in a clustered environment. So you might prefer to use the JPA based implementation which uses a database to store the message IDs which have been processed{snippet:id=idempotent|lang=java|url=camel/trunk/components/camel-jpa/src/test/java/org/apache/camel/processor/jpa/JpaIdempotentConsumerTest.java}In the above example we are using the header **messageId** to filter out duplicates and using the collection **myProcessorName** to indicate the Message ID Repository to use. This name is important as you could process the same message by many different processors; so each may require its own logical Message ID Repository.

For further examples of this pattern in use you could look at the junit test case

## Spring XML example

The following example will use the header **myMessageId** to filter out duplicates{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringIdempotentConsumerTest.xml}

## How to handle duplicate messages in the route

**Available as of Camel 2.8**

You can now set the `skipDuplicate` option to `false` which instructs the idempotent consumer to route duplicate messages as well. However the duplicate message has been marked as duplicate by having a property on the Exchange set to true. We can leverage this fact by using a Content Based Router or Message Filter to detect this and handle duplicate messages.

For example in the following example we use the Message Filter to send the message to a duplicate endpoint, and then stop continue routing that message.{snippet:id=e1|lang=java|title=Filter duplicate messages|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/IdempotentConsumerTest.java}The sample example in XML DSL would be:{snippet:id=e1|lang=xml|title=Filter duplicate messages|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringIdempotentConsumerNoSkipDuplicateFilterTest.xml}

## How to handle duplicate message in a clustered environment with a data grid

**Available as of Camel 2.8**

If you have running Camel in a clustered environment, a in memory idempotent repository doesn't work (see above). You can setup either a central database or use the idempotent consumer implementation based on the Hazelcast data grid. Hazelcast finds the nodes over multicast (which is default - configure Hazelcast for tcp-ip) and creates automatically a map based repository:

java HazelcastIdempotentRepository idempotentRepo = new HazelcastIdempotentRepository("myrepo"); from("direct:in").idempotentConsumer(header("messageId"), idempotentRepo).to("mock:out");

You have to define how long the repository should hold each message id (default is to delete it never). To avoid that you run out of memory you should create an eviction strategy based on the Hazelcast configuration. For additional information see camel-hazelcast.

See this little tutorial, how setup such an idempotent repository on two cluster nodes using Apache Karaf.

**Available as of Camel 2.13.0**

Another option for using Idempotent Consumer in a clustered environment is Infinispan. Infinispan is a data grid with replication and distribution clustering support. For additional information see **camel-infinispan.**

Using This Pattern