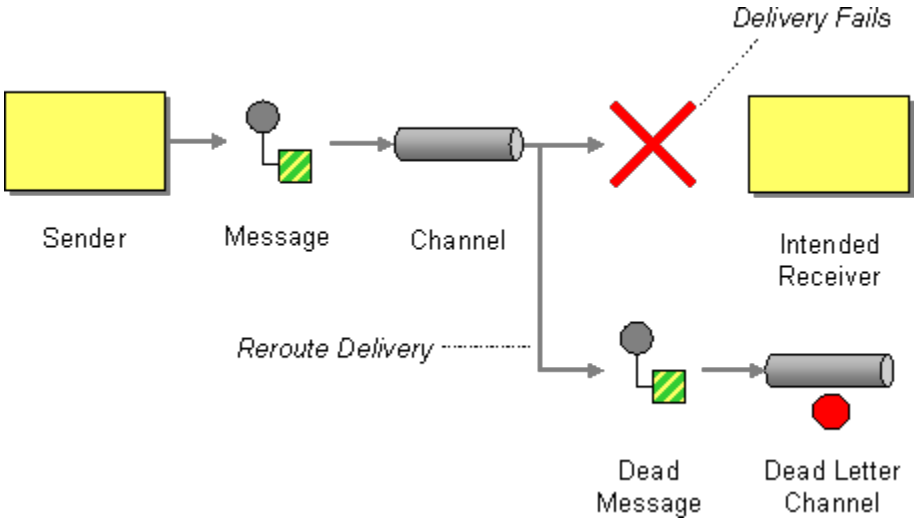
Dead Letter Channel

Dead Letter Channel

Camel supports the Dead Letter Channel from the EIP patterns using the DeadLetterChannel processor which is an Error Handler.



Differences Between The DeadLetterChannel And The DefaultErrorHandler
The **`DefaultErrorHandler`** does very little: it ends the Exchange immediately and propagates the thrown Exception back to the caller.

The **`DeadLetterChannel`** lets you control behaviors including redelivery, whether to propagate the thrown Exception to the caller (the **`handled`** option), and where the (failed) Exchange should now be routed to.

The **`DeadLetterChannel`** is also by default configured to not be verbose in the logs, so when a message is handled and moved to the dead letter endpoint, then there is nothing logged. If you want some level of logging you can use the various options on the redelivery policy / dead letter channel to configure this. For example if you want the message history then set **`logExhaustedMessageHistory=true`** (and **`logHandled=true`** for **Camel 2.15.x** or older).

When the **`DeadLetterChannel`** moves a message to the dead letter endpoint, any new Exception thrown is by default handled by the dead letter channel as well. This ensures that the **`DeadLetterChannel`** will always succeed. From **Camel 2.15**: this behavior can be changed by setting the option **`deadLetterHandleNewException=false`**. Then if a new Exception is thrown, then the dead letter channel will fail and propagate back that new Exception (which is the behavior of the default error handler). When a new Exception occurs then the dead letter channel logs this at **`WARN`** level. This can be turned off by setting **`logNewException=false`**.

Redelivery

It is common for a temporary outage or database deadlock to cause a message to fail to process; but the chances are if its tried a few more times with some time delay then it will complete fine. So we typically wish to use some kind of redelivery policy to decide how many times to try redeliver a message and how long to wait before redelivery attempts.

The RedeliveryPolicy defines how the message is to be redelivered. You can customize things like

- The number of times a message is attempted to be redelivered before it is considered a failure and sent to the dead letter channel.
- The initial redelivery timeout.
- Whether or not exponential backoff is used, i.e., the time between retries increases using a backoff multiplier.
- Whether to use collision avoidance to add some randomness to the timings.
- Delay pattern (see below for details).
- **Camel 2.11:** Whether to allow redelivery during stopping/shutdown.

Once all attempts at redelivering the message fails then the message is forwarded to the dead letter queue.

About Moving Exchange to Dead Letter Queue and Using `handled()`

`handled()` on Dead Letter Channel

When all attempts of redelivery have failed the Exchange is moved to the dead letter queue (the dead letter endpoint). The exchange is then complete and from the client point of view it was processed. As such the Dead Letter Channel have handled the Exchange.

For instance configuring the dead letter channel as:

Using the Fluent Builders

javaerrorHandler(deadLetterChannel("jms:queue:dead") .maximumRedeliveries(3).redeliveryDelay(5000));

Using the Spring XML Extensions

xml<route errorHandlerRef="myDeadLetterErrorHandler"> <!-- ... --> </route> <bean id="myDeadLetterErrorHandler" class="org.apache.camel.builder.DeadLetterChannelBuilder"> <property name="deadLetterUri" value="jms:queue:dead"/> <property name="redeliveryPolicy" ref="myRedeliveryPolicyConfig"/> </bean> <bean id="myRedeliveryPolicyConfig" class="org.apache.camel.processor.RedeliveryPolicy"> <property name="maximumRedeliveries" value="3"/> <property name="redeliveryDelay" value="5000"/> </bean>

The Dead Letter Channel above will clear the caused exception **`setException(null)`**, by moving the caused exception to a property on the Exchange, with the key **`Exchange.EXCEPTION_CAUGHT`**. Then the Exchange is moved to the **`jms:queue:dead`** destination and the client will not notice the failure.

About Moving Exchange to Dead Letter Queue and Using the Original Message

The option **`useOriginalMessage`** is used for routing the original input message instead of the current message that potentially is modified during routing.

For instance if you have this route:

java from("jms:queue:order:input") .to("bean:validateOrder") .to("bean:transformOrder") .to("bean:handleOrder");

The route listen for JMS messages and validates, transforms and handle it. During this the Exchange payload is transformed/modified. So in case something goes wrong and we want to move the message to another JMS destination, then we can configure our Dead Letter Channel with the **`useOriginalMessage`** option. But when we move the Exchange to this destination we do not know in which state the message is in. Did the error happen in before the **`transformOrder`** or after? So to be sure we want to move the original input message we received from **`jms:queue:order:input`**. So we can do this by enabling the **`useOriginalMessage`** option as shown below:

java// will use original body errorHandler(deadLetterChannel("jms:queue:dead") .useOriginalMessage() .maximumRedeliveries(5) .redeliverDelay(5000);

Then the messages routed to the **`jms:queue:dead`** is the original input. If we want to manually retry we can move the JMS message from the failed to the input queue, with no problem as the message is the same as the original we received.

OnRedelivery

When Dead Letter Channel is doing redeliver its possible to configure a Processor that is executed just **before** every redelivery attempt. This can be used for the situations where you need to alter the message before its redelivered. See below for sample.

onException and onRedeliver
We also support for per **onException** to set an **`onRedeliver`**. That means you can do special on redelivery for different exceptions, as opposed to **`onRedelivery`** set on Dead Letter Channel can be viewed as a global scope.

Redelivery Default Values

Redelivery is disabled by default.

The default redeliver policy will use the following values:

- **`maximumRedeliveries=0`**
- **`redeliverDelay=1000L`** (1 second)
- **`maximumRedeliveryDelay = 60 * 1000L`** (60 seconds)
- **`backOffMultiplier`** and **`useExponentialBackOff`** are ignored.
- **`retriesExhaustedLogLevel=LoggingLevel.ERROR`**
- **`retryAttemptedLogLevel=LoggingLevel.DEBUG`**
- Stack traces are logged for exhausted messages, from **Camel 2.2**.
- Handled exceptions are not logged, from **Camel 2.3**.
- **`logExhaustedMessageHistory`** is true for default error handler, and false for dead letter channel.
- **`logExhaustedMessageBody` Camel 2.17:** is disabled by default to avoid logging sensitive message body/header details. If this option is **`true`**, then **`logExhaustedMessageHistory`** must also be **`true`**.

The maximum redeliver delay ensures that a delay is never longer than the value, default 1 minute. This can happen when **`useExponentialBackOff=true`**.

The **`maximumRedeliveries`** is the number of **re**-delivery attempts. By default Camel will try to process the exchange 1 + 5 times. 1 time for the normal attempt and then 5 attempts as redeliveries.
Setting the **`maximumRedeliveries=-1`** (or < **`-1`**) will then always redelivery (unlimited).
Setting the **`maximumRedeliveries=0`** will disable re-delivery.

Camel will log delivery failures at the **`DEBUG`** logging level by default. You can change this by specifying **`retriesExhaustedLogLevel`** and/or **`retryAttemptedLogLevel`**. See ExceptionBuilderWithRetryLoggingLevelSetTest for an example.

You can turn logging of stack traces on/off. If turned off Camel will still log the redelivery attempt. It's just much less verbose.

Redeliver Delay Pattern

Delay pattern is used as a single option to set a range pattern for delays. When a delay pattern is in use the following options no longer apply:

- **`delay`**
- **`backOffMultiplier`**

- **useExponentialBackOff**
- **useCollisionAvoidance**
- **maximumRedeliveryDelay**

The idea is to set groups of ranges using the following syntax: **limit:delay;limit 2:delay 2;limit 3:delay 3;...;limit N:delay N**

Each group has two values separated with colon:

- **limit** = upper limit
- **delay** = delay in milliseconds
  And the groups is again separated with semi-colon. The rule of thumb is that the next groups should have a higher limit than the previous group.

Lets clarify this with an example:
delayPattern=5:1000;10:5000;20:20000

That gives us three groups:

- **5:1000**
- **10:5000**
- **20:20000**

Resulting in these delays between redelivery attempts:

- Redelivery attempt number **1..4 = 0ms** (as the first group start with 5)
- Redelivery attempt number **5..9 = 1000ms** (the first group)
- Redelivery attempt number **10..19 = 5000ms** (the second group)
- Redelivery attempt number **20.. = 20000ms** (the last group)

Note: The first redelivery attempt is **1**, so the first group should start with **1** or higher.

You can start a group with limit **1** to e.g., have a starting delay: **delayPattern=1:1000;5:5000**

- Redelivery attempt number **1..4 = 1000ms** (the first group)
- Redelivery attempt number **5.. = 5000ms** (the last group)

There is no requirement that the next delay should be higher than the previous. You can use any delay value you like. For example with **delayPattern=1:5000;3:1000** we start with 5 sec delay and then later reduce that to **1** second.

Redelivery header

When a message is redelivered the DeadLetterChannel will append a customizable header to the message to indicate how many times its been redelivered.
Before **Camel 2.6**: The header is **CamelRedeliveryCounter**, which is also defined on the **Exchange.REDELIVERY_COUNTER**.
From **Camel 2.6**: The header **CamelRedeliveryMaxCounter**, which is also defined on the **Exchange.REDELIVERY_MAX_COUNTER**, contains the maximum redelivery setting. This header is absent if you use **retryWhile** or have unlimited maximum redelivery configured.

And a boolean flag whether it is being redelivered or not (first attempt). The header **CamelRedelivered** contains a boolean if the message is redelivered or not, which is also defined on the **Exchange.REDELIVERED**.

Dynamically Calculated Delay From the Exchange

In **Camel 2.9** and **2.8.2**: The header is **CamelRedeliveryDelay**, which is also defined on the **Exchange.REDELIVERY_DELAY**. If this header is absent, normal redelivery rules apply.

Which Endpoint Failed

Available as of Camel 2.1

When Camel routes messages it will decorate the Exchange with a property that contains the **last** endpoint Camel send the Exchange to:

javaString lastEndpointUri = exchange.getProperty(Exchange.TO_ENDPOINT, String.class);

The **Exchange.TO_ENDPOINT** have the constant value **CamelToEndpoint**. This information is updated when Camel sends a message to any endpoint. So if it exists its the **last** endpoint which Camel send the Exchange to.

When for example processing the Exchange at a given Endpoint and the message is to be moved into the dead letter queue, then Camel also decorates the Exchange with another property that contains that **last** endpoint:

javaString failedEndpointUri = exchange.getProperty(Exchange.FAILURE_ENDPOINT, String.class);

The **Exchange.FAILURE_ENDPOINT** have the constant value **CamelFailureEndpoint**.

This allows for example you to fetch this information in your dead letter queue and use that for error reporting. This is usable if the Camel route is a bit dynamic such as the dynamic Recipient List so you know which endpoints failed.

Note: this information is retained on the Exchange even if the message is subsequently processed successfully by a given endpoint only to fail, for example, in local Bean processing instead. So, beware that this is a hint that helps pinpoint errors.

javafrom("activemq:queue:foo") .to("http://someserver/somepath") .beanRef("foo");

Now suppose the route above and a failure happens in the `foo` bean. Then the **`Exchange.TO_ENDPOINT`** and **`Exchange.FAILURE_ENDPOINT`** will still contain the value of `http://someserver/somepath`.

`OnPrepareFailure`

Available as of Camel 2.16

Before the exchange is sent to the dead letter queue, you can use **`onPrepare`** to allow a custom **`Processor`** to prepare the exchange, such as adding information why the Exchange failed.

For example, the following processor adds a header with the exception message:

java public static class MyPrepareProcessor implements Processor { @Override public void process(Exchange exchange) throws Exception { Exception cause = exchange.getProperty(Exchange.EXCEPTION_CAUGHT, Exception.class); exchange.getIn().setHeader("FailedBecause", cause.getMessage()); } }

Then configure the error handler to use the processor as follows:

javaerrorHandler(deadLetterChannel("jms:dead").onPrepareFailure(new MyPrepareProcessor()));

Configuring this from XML DSL is as follows:

xml<bean id="myPrepare" class="org.apache.camel.processor.DeadLetterChannelOnPrepareTest.MyPrepareProcessor"/> <errorHandler id="dlc" type="DeadLetterChannel" deadLetterUri="jms:dead" onPrepareFailureRef="myPrepare"/>

The **`onPrepare`** is also available using the default error handler.

Which Route Failed

Available as of Camel 2.10.4/2.11

When Camel error handler handles an error such as Dead Letter Channel or using Exception Clause with **`handled=true`**, then Camel will decorate the Exchange with the route id where the error occurred.

Example:

javaString failedRouteId = exchange.getProperty(Exchange.FAILURE_ROUTE_ID, String.class);

The **`Exchange.FAILURE_ROUTE_ID`** have the constant value **`CamelFailureRouteId`**. This allows for example you to fetch this information in your dead letter queue and use that for error reporting.

Control if Redelivery is Allowed During Stopping/Shutdown

Available as of Camel 2.11

Before **Camel 2.10**, Camel would perform redelivery while stopping a route, or shutting down Camel. This has improved a bit in **Camel 2.10**: Camel will no longer perform redelivery attempts when shutting down aggressively, e.g., during Graceful Shutdown and timeout hit.

From **Camel 2.11**: there is a new option **`allowRedeliveryWhileStopping`** which you can use to control if redelivery is allowed or not; notice that any in progress redelivery will still be executed. This option can only disallow any redelivery to be executed ***after*** the stopping of a route/shutdown of Camel has been triggered. If a redelivery is disallowed then a **`RejectedExcutionException`** is set on the Exchange and the processing of the Exchange stops. This means any consumer will see the Exchange as failed due the **`RejectedExcutionException`**. The default value is **`true`** for backward compatibility.

For example, the following snippet shows how to do this with Java DSL and XML DSL:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/RedeliveryErrorHandlerNoRedeliveryOnShutdownTest.java}And the sample sample with XML DSL{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringRedeliveryErrorHandlerNoRedeliveryOnShutdownTest.xml}

Samples

The following example shows how to configure the Dead Letter Channel configuration using the DSL{snippet:id=e3|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/ErrorHandlerTest.java}You can also configure the RedeliveryPolicy as this example shows{snippet:id=e4|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/ErrorHandlerTest.java}

How Can I Modify the Exchange Before Redelivery?

We support directly in Dead Letter Channel to set a Processor that is executed **before** each redelivery attempt. When Dead Letter Channel is doing redeliver its possible to configure a Processor that is executed just **before** every redelivery attempt. This can be used for the situations where you need to alter the message before its redelivered. Here we configure the Dead Letter Channel to use our processor **`MyRedeliveryProcessor`** to be executed before each redelivery.{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/DeadLetterChannelOnRedeliveryTest.java}And this is the processor **`MyRedeliveryProcessor`** where we alter the message.{snippet:id=e2|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/DeadLetterChannelOnRedeliveryTest.java}

How Can I Log What Caused the Dead Letter Channel to be Invoked?

You often need to know what went wrong that caused the Dead Letter Channel to be used and it does not offer logging for this purpose. So the Dead Letter Channel's endpoint can be set to a endpoint of our own (such as **`direct:deadLetterChannel`**). We write a route to accept this Exchange and log the Exception, then forward on to where we want the failed Exchange moved to (which might be a DLQ queue for instance). See also http://stackoverflow.com/questions/13711462/logging-camel-exceptions-and-sending-to-the-dead-letter-channel

Using This Pattern

- Error Handler
- Exception Clause