Error Handler

Error Handler

Camel supports pluggable ErrorHandler strategies to deal with errors processing an Event Driven Consumer. An alternative is to specify the error handling directly in the DSL using the Exception Clause.

For introduction and background material see Error handling in Camel.

Exception Clause
Using Error Handler combined with Exception Clause is a very powerful combination. We encourage end-users to use this combination in your error handling strategies. See samples and Exception Clause.
Using try ... catch ... finally
Related to error handling is the Try Catch Finally as DSL you can use directly in your route. Its basically a mimic of the regular try catch finally in the Java language but with more power.

The current implementations Camel provides out of the box are:

Non-transacted:

- DefaultErrorHandler is the default error handler in Camel. This error handler does not support a dead letter queue, it will propagate exceptions back to the caller, as if there where no error handler at all. It has a limited set of features.
- Dead Letter Channel which supports attempting to redeliver the message exchange a number of times before sending it to a dead letter endpoint
- **`LoggingErrorHandler`** for just catching and logging exceptions
- **`NoErrorHandler`** for no error handling

Transacted:

- TransactionErrorHandler is the default error handler in Camel for transacted routes. See the Transactional Client EIP pattern.

These error handlers can be applied in the DSL to an entire set of rules or a specific routing rule as we show in the next examples. Error handling rules are inherited on each routing rule within a single RouteBuilder

Short Summary of the Provided Error Handlers

DefaultErrorHandler

The DefaultErrorHandler is the default error handler in Camel. Unlike Dead Letter Channel it does not have any dead letter queue, and do **not** handle exceptions by default.

Dead Letter Channel

The Dead Letter Channel will redeliver at most 6 times using 1 second delay, and if the exchange failed it will be logged at **`ERROR`** level.

You can configure the default dead letter endpoint to use:{snippet:id=e3|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/ErrorHandlerTest.java}or in Spring DSL:

xml<bean id="deadLetterErrorHandler" class="org.apache.camel.builder.DeadLetterChannelBuilder"> <property name="deadLetterUri" value="log:dead"/> </bean> <camelContext errorHandlerRef="deadLetterErrorHandler" xmlns="http://camel.apache.org/schema/spring"> <!-- ... --> </camelContext>

or also from **Camel 2.3.0:**

xml<camel:errorHandler id="deadLetterErrorHandler" type="DeadLetterChannel" deadLetterUri="log:dead"> <camel:camelContext errorHandlerRef="deadLetterErrorHandler"> ... </camel:camelContext>

The `LoggingErrorHandler`

The logging error handler will log (by default at **`ERROR`** level) whenever an uncaught exception is thrown. The logging category, logger and level may all be defined in the builder.

javaerrorHandler(loggingErrorHandler("mylogger.name").level(LoggingLevel.INFO));

or in Spring DSL:

xml<bean id="loggingErrorHandler" class="org.apache.camel.builder.LoggingErrorHandler"> <property name="logName" value="mylogger.name"/> <property name="level" value="DEBUG"/> </bean> <camelContext errorHandlerRef="loggingErrorHandler" xmlns="http://camel.apache.org/schema/spring" > ... </camelContext>

or also from **Camel 2.3.0**:

xml<camel:errorHandler id="loggingErrorHandler" type="LoggingErrorHandler" logName="mylogger.name" level="DEBUG"/> <camel:camelContext errorHandlerRef="loggingErrorHandler"> ... </camel:camelContext>

This would create an error handler which logs exceptions using the category **`mylogger.name`** and uses the level **`INFO`** for all log messages created.

javafrom("seda:a") .errorHandler(loggingErrorHandler("mylogger.name").level(LoggingLevel.DEBUG)) .to("seda:b");

Loggers may also be defined for specific routes.

The `NoErrorHandler`

The no error handler is to be used for disabling error handling.

javaerrorHandler(noErrorHandler());

or in Spring DSL:

xml<bean id="noErrorHandler" class="org.apache.camel.builder.NoErrorHandlerBuilder"/> <camelContext errorHandlerRef="noErrorHandler" xmlns="http://camel.apache.org/schema/spring"> ... </camelContext>

or also from **Camel 2.3.0**:

xml<camel:errorHandler id="noErrorHandler" type="NoErrorHandler"/> <camel:camelContext errorHandlerRef="noErrorHandler"> ... </camel:camelContext>

TransactionErrorHandler

The TransactionErrorHandler is the default error handler in Camel for transacted routes.

[TransactionErrorHandler] is default for transacted routes
If you have marked a route as transacted using the **`transacted`** DSL then Camel will automatic use a TransactionErrorHandler. It will try to lookup the global/per route configured error handler and use it if its a **`TransactionErrorHandlerBuilder`** instance. If not Camel will automatic create a temporary TransactionErrorHandler that overrules the default error handler. This is convention over configuration.

Features Support by Various Error Handlers

Here is a breakdown of which features is supported by the Error Handler(s):

Feature	Supported By The Following Error Handler
`all scopes`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`onException`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`onWhen`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`continued`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`handled`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
Custom `ExceptionPolicy`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`useOriginalBody`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`retryWhile`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`onRedelivery`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`RedeliveryPolicy`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`asyncDelayedRedelivery`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`redeliverWhileStopping`	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
`dead letter queue`	Dead Letter Channel
`onPrepareFailure`	DefaultErrorHandler, Dead Letter Channel

See Exception Clause documentation for documentation of some of the features above.

Scopes

The error handler is scoped as either

- global
- per route

The following example shows how you can register a global error handler (in this case using the logging handler){snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/ErrorHandlerTest.java}The following example shows how you can register a route specific error handler; the customized logging handler is only registered for the route from Endpoint **`seda:a`**{snippet:id=e2|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/builder/ErrorHandlerTest.java}

Spring Based Configuration

Java DSL vs. Spring DSL
The error handler is configured a bit differently in Java DSL and Spring DSL. Spring DSL relies more on standard Spring bean configuration whereas Java DSL uses fluent builders.

The error handler can be configured as a spring bean and scoped in:

- global (the camelContext tag)
- per route (the route tag)
- or per policy (the policy/transacted tag)

The error handler is configured with the **`errorHandlerRef`** attribute.

Error Handler Hierarchy
The error handlers is inherited, so if you only have set a global error handler then its use everywhere. But you can override this in a route and use another error handler.

Spring Based Configuration Sample

In this sample we configure a Dead Letter Channel on the route that should redeliver at most 3 times and use a little delay before retrying.
First, we configure the reference to **`myDeadLetterErrorHandler`** using the **`errorHandlerRef`** attribute on the `route` tag.{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/config/DeadLetterChannelRedeliveryConfigTest-context.xml}Then we configure **`myDeadLetterErrorHandler`** that is our Dead Letter Channel. This configuration is standard Spring using the bean element.
Finally, we have another spring bean for the redelivery policy where we can configure the options for how many times to redeliver, delays etc.{snippet:id=e2|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/config/DeadLetterChannelRedeliveryConfigTest-context.xml}From **Camel 2.3.0**, camel provides a customer bean configuration for the Error Handler, you can find the examples here.{snippet:id=example|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/handler/ErrorHandlerDefinitionParser.xml}

Using the `TransactionalErrorHandler`

The **`TransactionalErrorHandler`** is based on spring transaction. This requires the usage of the camel-spring component. See Transactional Client that has many samples for how to use and transactional behavior and configuration with this error handler.

See also

- Error handling in Camel for introduction and background material on error handling in Camel
- Dead Letter Channel for the dead letter error handler
- DefaultErrorHandler for the default error handler in Camel
- TransactionErrorHandler for the default error handler for transacted routes
- Transactional Client for transactional behavior
- Exception Clause as it supports **handling** thrown exceptions
- Try Catch Finally for try ... catch ... finally as DSL you can use in the routing