Intercept

Intercept

The intercept feature in Camel supports intercepting Exchanges while they are *en route*. We have overhauled the Intercept in Camel 2.0 so the following information is based on Camel 2.0.

Camel supports three kinds of interceptors:

- **`intercept`** that intercepts each and every processing step while routing an Exchange in the route.
- **`interceptFrom`** that intercepts incoming Exchange in the route.
- **`interceptSendToEndpoint`** that intercepts when an Exchange is about to be sent to the given Endpoint.

These interceptors supports the following features:

- Predicate using `when` to only trigger the interceptor in certain conditions
- **`stop`** forces to stop continue routing the Exchange and mark it as completed successful. Camel will by default **not stop**.
- **`skip`** when used with **`interceptSendToEndpoint`** will **skip** routing the Exchange to the original endpoint. Camel will by default **not skip**.
- **`interceptFrom`** and **`interceptSendToEndpoint`** supports endpoint URI matching by: exact URI, wildcard, regular expression. See advanced section.
- The intercepted endpoint URI is stored as message header **`Exchange.INTERCEPTED_ENDPOINT`**.

stop
`stop` can be used in general, it does not have to be used with an Intercept you can use it in regular routing as well.

You can also instruct Camel to `stop` continue routing your message if you set the **`Exchange.ROUTE_STOP`** property to **`true`** or **`Boolean.TRUE`** on the Exchange. You can for instance do this from regular Java code in a Pojo or Processor.

`Intercept`

`Intercept` is like a regular interceptor that is applied on each processing step the Exchange undergo while its being routed. You can think of it as a *AOP before* that is applied at each DSL keyword you have defined in your route.

The classic Hello World example is:

intercept() .to("log:hello"); from("jms:queue:order") .to("bean:validateOrder") .to("bean:processOrder");

What happens is that the Exchange is intercepted before each processing step, that means that it will be intercepted before:

- `.to("bean:validateOrder")`
- `.to("bean:processOrder")`

So in this sample we intercept the Exchange twice.

The `when` predicate is also support on the **`intercept`** so we can attach a Predicate to only trigger the interception under certain conditions. For instance in the sample below we only intercept if the message body contains the string word **`Hello`**:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptSimpleRouteWhenTest.java}And in the route below we want to stop in certain conditions, when the message contains the word **`Hello`**:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptSimpleRouteWhenStopTest.java}

Using from Spring DSL

The same hello world sample in Spring DSL would be:

xml<camelContext> <intercept> <to uri="log:hello"/> </intercept> <route> <from uri="jms:queue:order"/> <to uri="bean:validateOrder"/> <to uri="bean:handleOrder"/> </route> </camelContext>

And the sample for using the **`when()`** predicate would be:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringInterceptSimpleRouteWhenTest.xml}And the sample for using the **`when()`** and **`stop()`** would be:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringInterceptSimpleRouteWhenStopTest.xml}

`InterceptFrom`

`InterceptFrom` is for intercepting any incoming Exchange, in any route (it intercepts all the `from` DSLs). This allows you to do some custom behavior for received Exchanges. You can provide a specific URI for a given Endpoint then it only applies for that particular route.

So lets start with the logging example. We want to log all the **incoming** requests so we use **`interceptFrom`** to route to the Log component. As `proceed` is default then the Exchange will continue its route, and thus it will continue to **`mock:first`**.{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptFromSimpleLogTest.java}You can also attach a Predicate to only trigger if certain conditions is meet. For instance in the route below we intercept when a test message is send to us, so we can do some custom processing before we continue routing:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptFromSimplePredicateTest.java}And if we want to filter out certain messages we can use the **`stop()`** to instruct Camel to stop continue routing the Exchange:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptFromSimplePredicateWithStopTest.java}And if want to only apply a specific endpoint, as the **`seda:bar`** endpoint in the sample below, we can do it like this:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptFromUriSimpleLogTest.java}

Using from Spring DSL

Intercept is of course also available using Spring DSL as shown in the sample below:{snippet:id=example|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringInterceptFromTest.xml}

Note: **`stop()`** is also supported in **`interceptFrom()`** so you can intercept from certain endpoints and route then elsewhere and **`stop()`** to not continue routing in the original intended route path.

InterceptSendToEndpoint

`InterceptSendToEndpoint`

Available as of Camel 2.0

Intercept send to endpoint is triggered when an Exchange is being sent to the intercepted endpoint. This allows you to route the Exchange to a Detour or do some custom processing before the Exchange is sent to the original intended destination. You can also skip sending to the intended destination. By default Camel will send to the original intended destination after the intercepted route completes. And as the regular intercept you can also define an `when` Predicate so we only intercept if the Predicate evaluates to **`true`**. This allows you do do a bit of filtering, to only intercept when certain criteria is meet.

Let start with a simple example, where we want to intercept when an Exchange is being sent to **`mock:foo`**:{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptSendToEndpointTest.java}And this time we add the Predicate so its only when the message body is **`Hello World`** we intercept.{snippet:id=e2|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptSendToEndpointTest.java}And to skip sending to the `mock:foo` endpoint we use the **`*skip()`** DSL in the route at the end to instruct Camel to skip sending to the original intended endpoint.{snippet:id=e3|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/intercept/InterceptSendToEndpointTest.java}

Conditional skipping
The combination of **`skipSendToEndpoint`** with a `when` predicate behaves differently depending on the Camel version:

- **Before Camel 2.10:** the skipping is applied unconditionally whether the **`when()`** predicate is matched or not, i.e. the **`when()`** predicate only determines whether the body of the interception will execute, but it does not control skipping behavior.
- **From Camel 2.10:** the skipping only occurs if the **`when()`** predicate is matched, leading to more natural logic altogether.

Using from Spring DSL

Intercept endpoint is of course also available using Spring DSL. We start with the first example from above in Spring DSL:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/interceptSendToEndpoint.xml}And the second. Notice how we can leverage the Simple language for the Predicate:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/interceptSendToEndpointWhen.xml}And the third with the **`skip`**; notice skip is set with the **`skipSendToOriginalEndpoint`** attribute on the **`interceptSendToEndpoint`** tag:{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/interceptSendToEndpointSkip.xml}

Advanced usage of `Intercept`

The **`interceptFrom`** and **`interceptSendToEndpoint`** supports endpoint URI matching by the following rules in the given order:

- match by exact URI name. This is the sample we have seen above.
- match by wildcard
- match by regular expression.

The real endpoint that was intercepted is stored as URI in the message IN header with the key **`Exchange.INTERCEPTED_ENDPOINT`**. This allows you to get hold of this information, when you for instance match by wildcard. Then you know the real endpoint that was intercepted and can react accordingly.

Match by Wildcard

Match by wildcard allows you to match a range of endpoint or all of a given type. For instance use **`uri="file:*"`** will match all File based endpoints:

javaintercept("jms:*") .to("log:fromjms");

Wildcards is match that the text before the **`*`** is matched against the given endpoint and if it also starts with the same characters its a match. For instance you can do:

javaintercept("file://order/inbox/*") .to("log:newfileorders");

To intercept any files received from the **`order/inbox`** folder.

Match by Regular Expression

Match by regular expression is just like match by wildcard but using regex instead. So if we want to intercept incoming messages from gold and silver JMS queues we can do:

javaintercept("jms:queue:(gold|silver)") .to("seda:handleFast"); About dynamic and static behavior of interceptFrom and interceptSendToEndpoint
The **`interceptSendToEndpoint`** is dynamic hence it will also trigger if a dynamic URI is constructed that Camel was not aware of at startup time.
The **`interceptFrom`** is not dynamic as it only intercepts input to routes registered as routes in **`CamelContext`**. So if you dynamic construct a `Consumer` using the Camel API and consumes an Endpoint then the **`interceptFrom`** is not triggered.

See Also

- Architecture
- AOP