**Dynamic Router**

The **Dynamic Router** from the EIP patterns allows you to route messages while avoiding the dependency of the router on all possible destinations while maintaining its efficiency.

In **Camel 2.5** we introduced a `dynamicRouter` in the DSL which is like a dynamic **Routing Slip** which evaluates the slip **on-the-fly**.

**Beware**

You must ensure the expression used for the `dynamicRouter` such as a bean, will return `null` to indicate the end. Otherwise the `dynamicRouter` will keep repeating endlessly.

**Options**

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uriDelimiter</code></td>
<td><code>,</code></td>
<td>Delimiter used if the <strong>Expression</strong> returned multiple endpoints.</td>
</tr>
<tr>
<td><code>ignoreInvalidIdEndpoints</code></td>
<td><code>false</code></td>
<td>If an endpoint uri could not be resolved, should it be ignored. Otherwise Camel will thrown an exception stating the endpoint uri is not valid.</td>
</tr>
<tr>
<td><code>cacheSize</code></td>
<td><code>1000</code></td>
<td><strong>Camel 2.13.1/2.12.4</strong>: Allows to configure the cache size for the <code>ProducerCache</code> which caches producers for reuse in the routing slip. Will by default use the default cache size which is 1000. Setting the value to <code>-1</code> allows to turn off the cache all together.</td>
</tr>
</tbody>
</table>

**Dynamic Router in Camel 2.5 onwards**

From Camel 2.5 the **Dynamic Router** will set a property (`Exchange.SLIP_ENDPOINT`) on the **Exchange** which contains the current endpoint as it advanced though the slip. This allows you to know how far we have processed in the slip. (It’s a slip because the **Dynamic Router** implementation is based on top of **Routing Slip**).

**Java DSL**

In Java DSL you can use the `dynamicRouter` as shown below:

```java
{snippet:id=e1|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/DynamicRouterTest.java}
```

Which will leverage a **Bean** to compute the slip **on-the-fly**, which could be implemented as follows:

```java
{snippet:id=e2|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/DynamicRouterTest.java}
```

Mind that this example is only for show and tell. The current implementation is not thread safe. You would have to store the state on the **Exchange**, to ensure thread safety, as shown below:

```java
{snippet:id=e2|lang=java|url=camel/trunk/camel-core/src/test/java/org/apache/camel/processor/DynamicRouterExchangePropertiesTest.java}
```

You could also store state as message headers, but they are not guaranteed to be preserved during routing, where as properties on the **Exchange** are. Although there was a bug in the method call expression, see the warning below.

Using beans to store state

Mind that in Camel 2.9.2 or older, when using a **Bean** the state is not propagated, so you will have to use a **Processor** instead. This is fixed in Camel 2.9.3 onwards.
**Spring XML**

The same example in Spring XML would be:

```xml
{snippet:id=e1|lang=xml|url=camel/trunk/components/camel-spring/src/test/resources/org/apache/camel/spring/processor/SpringDynamicRouterTest.xml}
```

**@DynamicRouter annotation**

You can also use the `@DynamicRouter` annotation, for example the Camel 2.4 example below could be written as follows. The `route` method would then be invoked repeatedly as the message is processed dynamically. The idea is to return the next endpoint uri where to go. Return `null` to indicate the end. You can return multiple endpoints if you like, just as the `Routing Slip`, where each endpoint is separated by a delimiter.

```java
public class MyDynamicRouter { @Consume(uri = "activemq:foo") @DynamicRouter public String route(@XPath("/customer/id") String customerId, @Header("Location") String location, Document body) { // query a database to find the best match of the endpoint based on the input parameters // return the next endpoint uri, where to go. Return null to indicate the end. }
}
```

**Dynamic Router in Camel 2.4 or older**

The simplest way to implement this is to use the `RecipientList Annotation` on a Bean method to determine where to route the message.

```java
public class MyDynamicRouter { @Consume(uri = "activemq:foo") @RecipientList public List<String> route(@XPath("/customer/id") String customerId, @Header("Location") String location, Document body) { // query a database to find the best match of the endpoint based on the input parameters ... }
}
```

In the above we can use the `Parameter Binding Annotations` to bind different parts of the `Message` to method parameters or use an `Expression` such as using `XPath` or `XQuery`.

The method can be invoked in a number of ways as described in the `Bean Integration` such as

- POJO Producing
- Spring Remoting
- Bean component

**Using This Pattern**