SpEL

Spring Expression Language (SpEL)

Available as of Camel 2.7

Camel allows SpEL to be used as an Expression or Predicate in the DSL or Xml Configuration.

Variables

The following variables are available in expressions and predicates written in SpEL:

Variable	Type	Description
this	Exchange	the Exchange is the root object
exchange	Exchange	the Exchange object
exception	Throwable	the Exchange exception (if any)
exchangeId	String	the exchange id
fault	Message	the Fault message (if any)
body	Object	**Camel 2.11:** The IN message body.
request	Message	the exchange.in message
response	Message	the exchange.out message (if any)
properties	Map	the exchange properties
property(name)	Object	the property by the given name
property(name, type)	Type	the property by the given name as the given type

Samples

Expression templating

SpEL expressions need to be surrounded by `#{ }` delimiters since expression templating is enabled. This allows you to combine SpEL expressions with regular text and use this as extremely lightweight template language.

For example if you construct the following route:

```
from("direct:example").setBody(spel("Hello #{request.body}! What a beautiful #{request.headers
['dayOrNight']}")).to("mock:result");
```

In the route above, notice spel is a static method which we need to import from `org.apache.camel.language.spel.SpelExpression.spel`, as we use spel as an Expression passed in as a parameter to the `setBody` method. Though if we use the fluent API we can do this instead:

```
from("direct:example").setBody().spel("Hello #{request.body}! What a beautiful #{request.headers
['dayOrNight']}").to("mock:result");
```

Notice we now use the `spel` method from the `setBody()` method. And this does not require us to static import the spel method from `org.apache.camel.language.spel.SpelExpression.spel`.

And sent a message with the string "World" in the body, and a header "dayOrNight" with value "day":

```
template.sendBodyAndHeader("direct:example", "World", "dayOrNight", "day");
```

The output on `mock:result` will be *"Hello World! What a beautiful day"*

Bean integration

You can reference beans defined in the Registry (most likely an `ApplicationContext`) in your SpEL expressions. For example if you have a bean named "foo" in your `ApplicationContext` you can invoke the "bar" method on this bean like this:

```
#{@foo.bar == 'xyz'}
```

SpEL in enterprise integration patterns

You can use SpEL as an expression for Recipient List or as a predicate inside a Message Filter:

```
<route>
  <from uri="direct:foo"/>
  <filter>
    <spel>#{request.headers['foo'] == 'bar'}</spel>
    <to uri="direct:bar"/>
  </filter>
</route>
```

And the equivalent in Java DSL:

```
   from("direct:foo").filter().spel("#{request.headers['foo'] == 'bar'}").to("direct:bar");
```

Loading script from external resource

Available as of Camel 2.11

You can externalize the script and have Camel load it from a resource such as `"classpath:"`, `"file:"`, or `"http:"`.
This is done using the following syntax: `"resource:scheme:location"`, eg to refer to a file on the classpath you can do:

```
.setHeader("myHeader").spel("resource:classpath:myspel.txt")
```

Dependencies

You need Spring 3.0 or higher to use Spring Expression Language. If you use Maven you could just add the following to your `pom.xml`:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring</artifactId>
  <version>xxx</version>
  <!-- use the same version as your Camel core version -->
</dependency>
```