How Do I Configure Endpoints?

How Do I Configure Endpoints?

There are a few different approaches to configuring components and endpoints.

Using Java

You can explicitly configure a Component using Java code as shown in this example

Or you can explicitly get hold of an Endpoint and configure it using Java code as shown in the Mock endpoint examples.

SomeEndpoint endpoint = camelContext.getEndpoint("someURI", SomeEndpoint.class); endpoint.setSomething("aValue");

Using CDI

You can use CDI as dependency injection framework to configure your Component or Endpoint instances.

For example, to configure the SJMS component, you can declare a producer method in a CDI bean:

javaclass MyCdiComponent { @PropertyInject("jms.maxConnections") int maxConnections; @Produces @Named("sjms") @ApplicationScoped SjmsComponent sjms() { SjmsComponent component = new SjmsComponent(); component.setConnectionFactory(new ActiveMQConnectionFactory ("vm://broker?broker.persistent=false")); component.setConnectionCount(maxConnections); return component; } }

Then, the component is lazily looked-up by Camel CDI whenever it is referenced, e.g., from the Camel Java DSL:

javaclass MyCdiRoute extends RouteBuilder { @Override public void configure() { from("sjms:sample.queue") .log("Received message [${body}]"); } }

Besides, endpoints of that component can be injected in any CDI beans, e.g.,

javaclass MyCdiBean { @Inject @Uri("sjms:sample.queue") Endpoint endpoint; }

Using Guice

You can also use Guice as the dependency injection framework. For example see the Guice JMS Example.

Using Spring XML

You can configure your Component or Endpoint instances in your Spring XML as follows.{snippet:id=example|lang=xml|url=camel/trunk/components/camel-jms/src/test/resources/org/apache/camel/component/jms/jmsRouteUsingSpring.xml}Which allows you to configure a component using some name (activemq in the above example), then you can refer to the component using **`activemq:[queue:|topic:]destinationName`**. This works by the **`SpringCamelContext`** lazily fetching components from the spring context for the scheme name you use for Endpoint URIs

Using Endpoint URIs

Another approach is to use the URI syntax. The URI syntax supports the query notation. So for example with the Mail component you can configure the password property via the URI

pop3://host:port?password=foo

Referencing Beans from Endpoint URIs

From Camel 2.0:

When configuring endpoints using URI syntax you can now refer to beans in the Registry using the **#** notation.
If the parameter value starts with a # sign then Camel will lookup in the Registry for a bean of the given type. For instance:

file://inbox?sorter=#mySpecialFileSorter

Will lookup a bean with the id **`mySpecialFileSorter`** in the Registry.

Configuring Parameter Values Using Raw Values, e.g., such as passwords

From Camel 2.11:

When configuring endpoint options using URI syntax, then the values is by default URI encoded. This can be a problem if you want to configure passwords and just use the value *as is* without any encoding. For example you may have a plus sign in the password, which would be decimal encoded by default.

From **Camel 2.11**: we made this easier as you can denote a parameter value to be **raw** using the following syntax **`RAW(value)`** e.g., the value starts with **`RAW(`** and then ends with the parenthesis **`)`**. Here is a little example:

.to("ftp:joe@myftpserver.com?password=RAW(se+re?t&23)&binary=true"

In the above example, we have declare the password value as raw, and the actual password would be as typed, e.g., **`se+re?t&23`**.

Using Property Placeholders

Camel have extensive support for using property placeholders, which you can read more about here. For example in the ftp example above we can externalize the password to a **`.properties`** file.

For example configuring the property placeholder when using a XML DSL, where we declare the location of the .properties file. Though we can also define this in Java code. See the documentation for more details.

xml<camelContext ...> <propertyPlaceholder id="properties" location="myftp.properties"/> ... </camelContext>

And the Camel route now refers to the placeholder using the **{{** key **}}** notation:

java.to("ftp:joe@myftpserver.com?password={{myFtpPassword}}&binary=true"

And have a **`myftp.properties`** file with password. Notice we still define the RAW(value) style to ensure the password is used *as is*

myFtpPassword=RAW(se+re?t&23)

We could still have used the RAW(value) in the Camel route instead:

java.to("ftp:joe@myftpserver.com?password=RAW({{myFtpPassword}})&binary=true"

And then we would need to remove the RAW from the properties file:

myFtpPassword=se+re?t&23

To understand more about property placeholders, read the documentation.

Configuring URIs using Endpoint with Bean Property Style

Available as of Camel 2.15

Sometimes configuring endpoint uris may have many options, and therefore the URI can become long. In Java DSL you can break the URIs into new lines as its just Java code, e.g., just concat the String. When using XML DSL then the URI is an attribute, e.g., **`<from uri="bla bla"/>`**. From **Camel 2.15**: you can configure the endpoint separately, and from the routes refer to the endpoints using their shorthand IDs.

xml<camelContext ...> <endpoint id="foo" uri="ftp://foo@myserver"> <property name="password" value="secret"/> <property name="recursive" value="true"/> <property name="ftpClient.dataTimeout" value="30000"/> <property name="ftpClient.serverLanguageCode" value="fr"/> </endpoint> <route> <from uri="ref:foo"/> ... </route> </camelContext>

In the example above, the endpoint with id foo, is defined using **`<endpoint>`** which under the covers assembles this as an URI, with all the options, as if you have defined all the options directly in the URI. You can still configure some options in the URI, and then use **`<property>`** style for additional options, or to override options from the URI, such as:

xml<endpoint id="foo" uri="ftp://foo@myserver?recursive=true"> <property name="password" value="secret"/> <property name="ftpClient.dataTimeout" value="30000"/> <property name="ftpClient.serverLanguageCode" value="fr"/> </endpoint>

Configuring Long URIs Using New Lines

Available as of Camel 2.15

Sometimes configuring endpoint URIs may have many options, and therefore the URI can become long. In Java DSL you can break the URIs into new lines as its just Java code, e.g., just concat the String. When using XML DSL then the URI is an attribute, e.g., **`<from uri="bla bla"/>`**. From **Camel 2.15**: you can break the URI attribute using new line, such as shown below:

xml<route> <from uri="ftp://foo@myserver?password=secret& recursive=true& ftpClient.dataTimeout=30000& ftpClientConfig.serverLanguageCode=fr"/> <to uri="bean:doSomething"/> </route>

Notice that it still requires to use escape **`&`** as **`&`** in XML. Also you can have multiple options in one line, e.g., this is the same:

xml<route> <from uri="ftp://foo@myserver?password=secret& recursive=true&ftpClient.dataTimeout=30000& ftpClientConfig.serverLanguageCode=fr"/> <to uri="bean:doSomething"/> </route>

See Also

- How do I add a component
- Spring
- URIs
- Using PropertyPlaceholder