Jasypt

Jasypt component

Available as of Camel 2.5

Jasypt is a simplified encryption library which makes encryption and decryption easy. Camel integrates with Jasypt to allow sensitive information in Properties files to be encrypted. By dropping **`camel-jasypt`** on the classpath those encrypted values will automatically be decrypted on-the-fly by Camel. This ensures that human eyes can't easily spot sensitive information such as usernames and passwords.

Maven users will need to add the following dependency to their `pom.xml` for this component:

xml<dependency> <groupId>org.apache.camel</groupId> <artifactId>camel-jasypt</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency>

Tooling

The Jasypt component provides a little command line tooling to encrypt or decrypt values.

The console output the syntax and which options it provides:

Apache Camel Jasypt takes the following options -h or -help = Displays the help screen -c or -command <command> = Command either encrypt or decrypt -p or -password <password> = Password to use -i or -input <input> = Text to encrypt or decrypt -a or -algorithm <algorithm> = Optional algorithm to use

For example to encrypt the value `tiger` you run with the following parameters. In the apache camel kit, you cd into the lib folder and run the following java cmd, where *<CAMEL_HOME>* is where you have downloaded and extract the Camel distribution.

$ cd <CAMEL_HOME>/lib $ java -jar camel-jasypt-2.5.0.jar -c encrypt -p secret -i tiger

Which outputs the following result

Encrypted text: qaEEacuW7BUti8LcMgyjKw==

This means the encrypted representation `qaEEacuW7BUti8LcMgyjKw==` can be decrypted back to `tiger` if you know the master password which was `secret`.
If you run the tool again then the encrypted value will return a different result. But decrypting the value will always return the correct original value.

So you can test it by running the tooling using the following parameters:

$ cd <CAMEL_HOME>/lib $ java -jar camel-jasypt-2.5.0.jar -c decrypt -p secret -i qaEEacuW7BUti8LcMgyjKw==

Which outputs the following result:

Decrypted text: tiger

The idea is then to use those encrypted values in your Properties files. Notice how the password value is encrypted and the value has the tokens surrounding `ENC(value here)`

{snippet:id=e1|lang=java|url=camel/trunk/components/camel-jasypt/src/test/resources/org/apache/camel/component/jasypt/myproperties.properties}

Tooling dependencies for Camel 2.5 and 2.6

The tooling requires the following JARs in the classpath, which has been enlisted in the `MANIFEST.MF` file of `camel-jasypt` with `optional/` as prefix. Hence why the java cmd above can pickup the needed JARs from the Apache Distribution in the `optional` directory.

jasypt-1.6.jar commons-lang-2.4.jar commons-codec-1.4.jar icu4j-4.0.1.jar Java 1.5 users
The `icu4j-4.0.1.jar` is only needed when running on JDK 1.5.

This JAR is not distributed by Apache Camel and you have to download it manually and copy it to the `lib/optional` directory of the Camel distribution. You can download it from Apache Central Maven repo.

Tooling dependencies for Camel 2.7 or better

Jasypt 1.7 onwards is now fully standalone so no additional JARs is needed.

URI Options

The options below are exclusive for the Jasypt component.

confluenceTableSmall

Name	Default Value	Type	Description

password	null	String	Specifies the master password to use for decrypting. This option is mandatory. See below for more details.
algorithm	null	String	Name of an optional algorithm to use.

Protecting the master password

The master password used by Jasypt must be provided, so that it's capable of decrypting the values. However having this master password out in the open may not be an ideal solution. Therefore you could for example provide it as a JVM system property or as a OS environment setting. If you decide to do so then the password option supports prefixes which dictates this. sysenv: means to lookup the OS system environment with the given key. sys: means to lookup a JVM system property.

For example you could provided the password before you start the application

$ export CAMEL_ENCRYPTION_PASSWORD=secret

Then start the application, such as running the start script.

When the application is up and running you can unset the environment

$ unset CAMEL_ENCRYPTION_PASSWORD

The password option is then a matter of defining as follows: password=sysenv:CAMEL_ENCRYPTION_PASSWORD.

Example with Java DSL

In Java DSL you need to configure Jasypt as a JasyptPropertiesParser instance and set it on the Properties component as show below:

{snippet:id=e1|lang=java|url=camel/trunk/components/camel-jasypt/src/test/java/org/apache/camel/component/jasypt/JasyptPropertiesTest.java}

The properties file myproperties.properties then contain the encrypted value, such as shown below. Notice how the password value is encrypted and the value has the tokens surrounding ENC(value here)

{snippet:id=e1|lang=java|url=camel/trunk/components/camel-jasypt/src/test/resources/org/apache/camel/component/jasypt/myproperties.properties}

Example with Spring XML

In Spring XML you need to configure the JasyptPropertiesParser which is shown below. Then the Camel Properties component is told to use jasypt as the properties parser, which means Jasypt has its chance to decrypt values looked up in the properties.

xml<!-- define the jasypt properties parser with the given password to be used --> <bean id="jasypt" class="org.apache.camel.component.jasypt. JasyptPropertiesParser"> <property name="password" value="secret"/> </bean> <!-- define the camel properties component --> <bean id="properties" class="org.apache.camel.component.properties.PropertiesComponent"> <!-- the properties file is in the classpath --> <property name="location" value=" classpath:org/apache/camel/component/jasypt/myproperties.properties"/> <!-- and let it leverage the jasypt parser --> <property name="propertiesParser" ref="jasypt"/> </bean>

The Properties component can also be inlined inside the <camelContext> tag which is shown below. Notice how we use the propertiesParserRef attribute to refer to Jasypt.

<!-- define the jasypt properties parser with the given password to be used --> <bean id="jasypt" class="org.apache.camel.component.jasypt. JasyptPropertiesParser"> <!-- password is mandatory, you can prefix it with sysenv: or sys: to indicate it should use an OS environment or JVM system property value, so you dont have the master password defined here --> <property name="password" value="secret"/> </bean> <camelContext xmlns=" http://camel.apache.org/schema/spring"> <!-- define the camel properties placeholder, and let it leverage jasypt --> <propertyPlaceholder id="properties" location="classpath:org/apache/camel/component/jasypt/myproperties.properties" propertiesParserRef="jasypt"/> <route> <from uri="direct:start"/> <to uri="{{cool.result}}"/> </route> </camelContext>

Example with Blueprint XML

In Blueprint XML you need to configure the JasyptPropertiesParser which is shown below. Then the Camel Properties component is told to use jasy pt as the properties parser, which means Jasypt has its chance to decrypt values looked up in the properties.

xml<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cm="http://aries. apache.org/blueprint/xmlns/blueprint-cm/v1.0.0" xsi:schemaLocation=" http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0 /blueprint.xsd"> <cm:property-placeholder id="myblue" persistent-id="mypersistent"> <!-- list some properties for this test --> <cm:default-properties> <cm: property name="cool.result" value="mock:{{cool.password}}"/> <cm:property name="cool.password" value="ENC(bsW9uV37gQ0QHFu7KO03Ww==)"/> < /cm:default-properties> </cm:property-placeholder> <!-- define the jasypt properties parser with the given password to be used --> <bean id="jasypt" class="org.apache.camel.component.jasypt.JasyptPropertiesParser"> <property name="password" value="secret"/> </bean> <camelContext xmlns=" http://camel.apache.org/schema/blueprint"> <!-- define the camel properties placeholder, and let it leverage jasypt --> <propertyPlaceholder id="properties" location="blueprint:myblue" propertiesParserRef="jasypt"/> <route> <from uri="direct:start"/> <to uri="{{cool.result}}"/> </route> </camelContext> < /blueprint>

The Properties component can also be inlined inside the <camelContext> tag which is shown below. Notice how we use the propertiesParserRef at tribute to refer to Jasypt.

xml<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cm="http://aries. apache.org/blueprint/xmlns/blueprint-cm/v1.0.0" xsi:schemaLocation=" http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0 /blueprint.xsd"> <!-- define the jasypt properties parser with the given password to be used --> <bean id="jasypt" class="org.apache.camel.component. jasypt.JasyptPropertiesParser"> <property name="password" value="secret"/> </bean> <camelContext xmlns="http://camel.apache.org/schema/blueprint"

> <!-- define the camel properties placeholder, and let it leverage jasypt --> <propertyPlaceholder id="properties" location="classpath:org/apache/camel /component/jasypt/myproperties.properties" propertiesParserRef="jasypt"/> <route> <from uri="direct:start"/> <to uri="{{cool.result}}"/> </route> < /camelContext> </blueprint>

See Also

- Security
- Properties
- Encrypted passwords in ActiveMQ - ActiveMQ has a similar feature as this `camel-jasypt` component