There are **two different ways** to send messages to any Camel [Endpoint](http://camel.apache.org/endpoint.html) from a POJO

**1.By Using @EndpointInject**

To allow sending of messages from POJOs you can use the [@EndpointInject](http://camel.apache.org/maven/current/camel-core/apidocs/org/apache/camel/EndpointInject.html) annotation. This will inject a [ProducerTemplate](http://camel.apache.org/maven/current/camel-core/apidocs/org/apache/camel/ProducerTemplate.html) so that the bean can participate in message exchanges.

Example: send a message to the **foo.bar** ActiveMQ queue:

|  |
| --- |
| public class Foo {    @EndpointInject(uri="activemq:foo.bar")    ProducerTemplate producer;      public void doSomething() {      if (whatever) {        producer.sendBody("<hello>world!</hello>");      }    }  } |

The downside of this is that your code is now dependent on a Camel API, the **ProducerTemplate**. The next section describes how to remove this dependency.

See [POJO Consuming](http://camel.apache.org/pojo-consuming.html) for how to use a property on the bean as endpoint configuration, e.g., using the **property** attribute on **@Produce**, **@EndpointInject**.

**2.By Hiding the Camel APIs From Your Code Using @Produce**

We recommend [Hiding Middleware](http://camel.apache.org/hiding-middleware.html) APIs from your application code so the next option might be more suitable. You can add the **@Produce** annotation to an injection point (a field or property setter) (1)using a **ProducerTemplate or** using (2)some interface you use in your business logic. Example:

|  |
| --- |
| public interface MyListener {      String sayHello(String name);  }    public class MyBean {      @Produce(uri = "activemq:foo")      protected MyListener producer;        public void doSomething() {          // lets send a message          String response = producer.sayHello("James");      }  } |

Here Camel will automatically inject a smart client side proxy at the **@Produce** annotation - an instance of the **MyListener** instance. When we invoke methods on this interface the method call is turned into an object and using the Camel [Spring Remoting](http://camel.apache.org/spring-remoting.html) mechanism it is sent to the endpoint - in this case the [ActiveMQ](http://camel.apache.org/activemq.html) endpoint to queue **foo**; then the caller blocks for a response.

If you want to make asynchronous message sends then use [an @InOnly annotation on the injection point](http://camel.apache.org/using-exchange-pattern-annotations.html).

<http://camel.apache.org/pojo-producing.html>

## Bean Integration

Camel supports the integration of beans and POJOs in a number of ways

### Annotations

If a bean is defined in [Spring](http://camel.apache.org/spring.html) XML or scanned using the Spring component scanning mechanism and a **<camelContext>** is used or a CamelBeanPostProcessor then we process a number of Camel annotations to do various things such as injecting resources or producing, consuming or routing messages.

The following annotations is supported and inject by Camel's CamelBeanPostProcessor

|  |  |
| --- | --- |
| Annotation | Description |
| @EndpointInject | To inject an endpoint, see more details at [POJO Producing](http://camel.apache.org/pojo-producing.html). |
| @BeanInject | **Camel 2.13:** To inject a bean obtained from the [Registry](http://camel.apache.org/registry.html). See [Bean Injection](http://camel.apache.org/bean-injection.html). |
| @PropertyInject | **Camel 2.12:** To inject a value using [property placeholder](http://camel.apache.org/using-propertyplaceholder.html). |
| @Produce | To inject a producer to send message to an endpoint. See [POJO Producing](http://camel.apache.org/pojo-producing.html). |
| @Consume | To inject a consumer on a method. See [POJO Consuming](http://camel.apache.org/pojo-consuming.html). |

<http://camel.apache.org/bean-integration.html>

**@Consume**

To consume a message you use the [@Consume](http://camel.apache.org/maven/current/camel-core/apidocs/org/apache/camel/Consume.html) annotation to mark a particular method of a bean as being a **consumer** method. The uri of the annotation defines the Camel [Endpoint](http://camel.apache.org/endpoint.html) to consume from.

e.g. lets invoke the onCheese() method with the String body of the inbound JMS message from [ActiveMQ](http://camel.apache.org/activemq.html) on the cheese queue; this will use the [Type Converter](http://camel.apache.org/type-converter.html) to convert the JMS ObjectMessage or BytesMessage to a String - or just use a TextMessage from JMS

|  |
| --- |
| public class Foo {      @Consume(uri="activemq:cheese")    public void onCheese(String name) {      ...    }  } |

The [Bean Binding](http://camel.apache.org/bean-binding.html) is then used to convert the inbound [Message](http://camel.apache.org/message.html) to the parameter list used to invoke the method .

What this does is basically create a route that looks kinda like this

|  |
| --- |
| from(uri).bean(theBean, "methodName"); |

<http://camel.apache.org/pojo-consuming.html>