# POJO Producing

There are two different ways to send messages to any Camel Endpoint from a POJO

### `@EndpointInject`

To allow sending of messages from POJOs you can use the @EndpointInject annotation. This will inject a ProducerTemplate 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 for how to use a property on the bean as endpoint configuration, e.g., using the **`property`** attribute on **`@Produce`**, **`@EndpointInject`**.

## Hiding the Camel APIs From Your Code Using `@Produce`

We recommend Hiding Middleware 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) using a **`ProducerTemplate`** **or** using 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 mechanism it is sent to the endpoint - in this case the ActiveMQ 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.