

# Data Format

## Data Format

Camel supports a pluggable DataFormat to allow messages to be marshalled to and from binary or text formats to support a kind of [Message Translator](#).

The following data formats are currently supported:

- Standard JVM object marshalling
  - [Serialization](#)
  - [String](#)
- Object marshalling
  - [Avro](#)
  - [Boon](#)
  - [Hessian](#)
  - [JSON](#)
  - [Protobuf](#)
  - [YAML](#)
- Object/XML marshalling
  - [Castor](#)
  - [JAXB](#)
  - [XmlBeans](#)
  - [XStream](#)
  - [JiBX](#)
  - [Jackson XML](#)
- Object/XML/Webservice marshalling
  - [SOAP](#)
- Direct JSON / XML marshalling
  - [XmlJson](#)
- Flat data structure marshalling
  - [BeanIO](#)
  - [Bindy](#)
  - [CSV](#)
  - [EDI](#)
  - [Flatpack DataFormat](#)
  - [uniVocity-parsers formats](#)
- Domain specific marshalling
  - [HL7 DataFormat](#)
- Compression
  - [GZip data format](#)
  - [Zip DataFormat](#)
  - [Zip File DataFormat](#)
  - [LZF Data Format](#)
  - [Tar DataFormat](#)
- Security
  - [Crypto](#)
  - [PGP](#)
  - [XMLSecurity DataFormat](#)
- Misc.
  - [Base64](#)
  - [Custom DataFormat](#) - to use your own custom implementation
  - [MIME-Multipart](#)
  - [RSS](#)
  - [TidyMarkup](#)
  - [Syslog](#)
  - [iCal](#)
  - [Barcode](#) - to read and generate barcodes (QR-Code, PDF417, ...)

And related is the following:

- [DataFormat Component](#) for working with [Data Formats](#) as if it was a regular [Component](#) supporting [Endpoints](#) and [URIs](#).
- [Dozer Type Conversion](#) using Dozer for type converting POJOs

## Unmarshalling

If you receive a message from one of the Camel [Components](#) such as [File](#), [HTTP](#) or [JMS](#) you often want to unmarshal the payload into some bean so that you can process it using some [Bean Integration](#) or perform [Predicate](#) evaluation and so forth. To do this use the **unmarshal** word in the [DSL](#) in Java or the [Xml Configuration](#).

For example

```
DataFormat jaxb = new JaxbDataFormat("com.acme.model");

from("activemq:My.Queue").
    unmarshal(jaxb).
    to("mqseries:Another.Queue");
```

The above uses a named DataFormat of *jaxb* which is configured with a number of Java package names. You can if you prefer use a named reference to a data format which can then be defined in your [Registry](#) such as via your [Spring XML](#) file.

You can also use the DSL itself to define the data format as you use it. For example the following uses Java serialization to unmarshal a binary file then send it as an [ObjectMessage](#) to [ActiveMQ](#)

```
from("file://foo/bar").
    unmarshal().serialization().
    to("activemq:Some.Queue");
```

## Marshalling

Marshalling is the opposite of unmarshalling, where a bean is marshalled into some binary or textual format for transmission over some transport via a Camel [Component](#). Marshalling is used in the same way as unmarshalling above; in the [DSL](#) you can use a DataFormat instance, you can configure the DataFormat dynamically using the DSL or you can refer to a named instance of the format in the [Registry](#).

The following example unmarshals via serialization then marshals using a named JAXB data format to perform a kind of [Message Translator](#)

```
from("file://foo/bar").
    unmarshal().serialization().
    marshal("jaxb").
    to("activemq:Some.Queue");
```

## Using Spring XML

This example shows how to configure the data type just once and reuse it on multiple routes

Error rendering macro 'code': Invalid value specified for parameter 'java.lang.NullPointerException'

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <dataFormats>
    <jaxb id="myJaxb" prettyPrint="true" contextPath="org.apache.camel.example" />
  </dataFormats>

  <route>
    <from uri="direct:start" />
    <marshal><custom ref="myJaxb" /></marshal>
    <to uri="direct:marshalled" />
  </route>
  <route>
    <from uri="direct:marshalled" />
    <unmarshal><custom ref="myJaxb" /></unmarshal>
    <to uri="mock:result" />
  </route>
</camelContext>
```

You can also define reusable data formats as Spring beans

```
<bean id="myJaxb" class="org.apache.camel.model.dataformat.JaxbDataFormat">  
  <property name="prettyPrint" value="true"/>  
  <property name="contextPath" value="org.apache.camel.example"/>  
</bean>
```