Juffrou Reference Documentation

2.1.8-SNAPSHOT

Carlos Martins

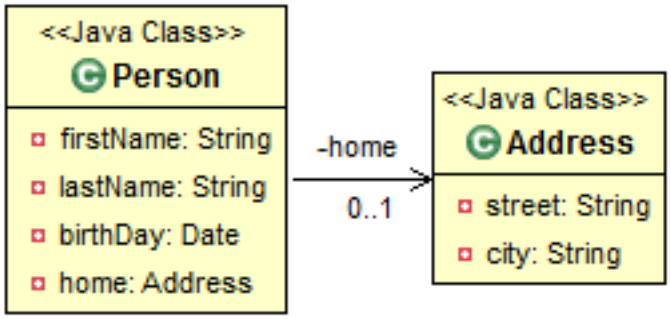
Juffrou Reference Documentation: 2.1.8-SNAPSHOT

Carlos Martins

This document refers to version 2.1.8-SNAPSHOT of the Juffrou group of libraries.

Chapter 1. Introduction

Juffrou stands for Java Utilities Framework For the Rest Of Us and is a collection of useful classes or mini frameworks to help the java developer.




```
getValue("firstName")
getValue("home.city")
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Person>
  <firstName>Carlos</firstName>
  <lastName>Martins</lastName>
  <birthDay>1967-10-01</birthDay>
  <homeTown>Lisboa</homeTown>
</Person>
```

Do we need another XML marshalling / unmarshalling framework when there are already so many?

Well, it's true that there are many XML marshalling frameworks. I have used extensively XStream and Castor and I have huge admiration for those two frameworks. I am also a huge fan of cowtowncoder and his amazing jackson and fasterXml projects.

But as I got to know those frameworks I saw the good and the not so good in them and was always left with a feeling that there is something lacking.

I wanted something focused on marshalling java beans. Easy to configure with a choice for mapping file configuration or configuration through code. I wanted it to be easy to use, flexible - easy to extend and something that would handle the marshalling of nested beans into flat XML and back without pain.

Juffrou-XML is a first in that. So yes - we need this XML marshalling / unmarshalling framework.

Chapter 2. Getting Started with Juffrou-reflect

Introduction

Juffrou-reflect is focused on reflection, and offers a very performant bean wrapper allowing bean introspection and manipulation through property names.

Installing

Maven projects

To start using Juffrou-XML in your maven project just add the following dependency:

```
<dependency>
 <groupId>net.sf.juffrou</groupId>
 <artifactId>juffrou-reflect</artifactId>
 <version>2.1.8-SNAPSHOT</version>
</dependency>
```

This will allow you access the source code of the library as well as the javadoc files, if you have checked the options "download artifact sources" and "download atifact javadoc in your IDE."

Non maven projects

Download the file `juffrou-2.1.8-SNAPSHOT-bundle.zip` from the *website* [http://cemartins.github.io/juffrou/index.html] and extract it's contents to a temporary directory.

Add `juffrou-reflect-2.1.8-SNAPSHOT.jar` to the classpath of your project and you are good to go.

Chapter 3. Using Juffrou-reflect

JuffrouBeanWrapper

JuffrouBeanWrapper is the object that wraps around your beans and allows you to inspect them through the names of their properties. You can access the wrapped bean's properties and also the properties of beans referenced by them.

Figure 3.1. Example class diagram

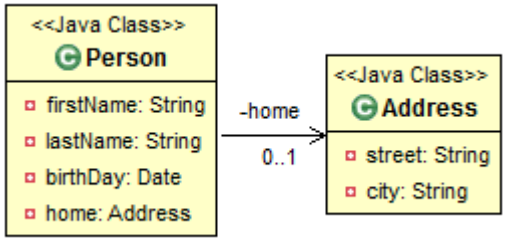


Figure 3.1, “Example class diagram” shows one class Person with three simple attributes and one attribute of type Address. The Address class is also shown and has two simple attributes.
Person and Address will be used extensively throughout this manual in code examples.
A tipical use of JuffrouBeanWrapper could be:

```
JuffrouBeanWrapper bw = new JuffrouBeanWrapper(Person.class); ❶
bw.setValue("firstName", "Carlos"); ❷
bw.setValue("home.city", "Lisboa"); ❸
Person person = (Person) bw.getBean(); ❹
```

Note: you can create a JuffrouBeanWrapper around a class or around an object instance.

❶ Instantiate JuffrouBeanWrapper around a Person class
❷ Set the firstName property of Person with the value "Carlos".
❸ Set the home property of Person with a new instance of an Address class and set the city property of Address to the value "Lisboa".
❹ Get the instance of the wrapped object.

When the program executes `bw.setValue("firstName", "Carlos");` the bean wrapper creates an instance of Person. And when it executes `bw.setValue("home.city", "Lisboa");` it will instantiate an Address class, set the value of the property `city` in Address to "Lisboa" and set the value of the property `home` in Person to the created Address instance.

Setting Bean Property Values

When you execute the method `bw.setValue("birthDay", someObject)` to set the value of the birthDay property, the JuffrouBeanWrapper expects `someObject` to be of the same type as the bean property (in this case Date). And if it is not, it will throw an IllegalArgumentException exception.

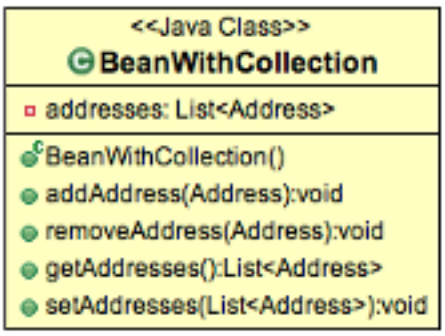
But if you use the method `setValueOfString("birthDay", "1967-10-01")`, then the JuffrouBeanWrapper will try to convert the string value into the type of the property before setting it.

Collection properties

The methods `addElement(String propertyName, Object element)` and `removeElement(String propertyName, Object element)` allow you to add and remove elements from a bean property of type collection.

These methods will add/remove the element directly from the collection property or call a method in the underlying bean to do the job.

Figure 3.2. Example bean with a collection property



If you have a bean with a property like `List<Address> addresses` and that bean contains the methods `addAddress(Address address)` and `removeAddress(Address address)`, then these methods will be used to add remove elements from the collection. If not, then the normall collection methods will be used. In this case, the method `addElement(propertyName, element)` will be the equivalent of `((Collection)getValue(propertyName)).add(element)`.

Nested BeanWrappers

When you access a nested property (i.e. a property of a nested bean), like in the case of `bw.setValue("home.city", "Lisboa")`, the JuffrouBeanWrapper automatically creates another JuffrouBeanWrapper around the nested bean. In this case around an Address bean. This is called a nested bean wrapper.

Nested JuffrouBeanWrappers are created only when referenced by `beanWrapper.getValue`, `beanWrapper.setValue`, `beanWrapper.getType`, `beanWrapper.getClass` or `beanWrapper.getNestedWrapper`.

You can get a specific nested bean wrapper with the method `beanWrapper.getNestedWrapper("home")` and you can get all current nested bean wrappers with the method `beanWrapper.getNestedWrappers()`.

If you want to change the wrapped instance without creating a new JuffrouBeanWrapper you can call `bw.setBean(newInstance)` and if you want to zero all properties of the wrapped instance you can call `bw.setBean(null)`.

In case you don't know the details of the Wrapped object, you can inquire the JuffrouBeanWrapper:

Example 3.1. Inquiring the JuffrouBeanWrapper:

```
BeanWrapperContext context = BeanWrapperContext.create(Programmer.class);
JuffrouBeanWrapper beanWrapper = new JuffrouBeanWrapper(context);
for(String propertyName : beanWrapper.getPropertyNames()) {
 Type type = beanWrapper.getType(propertyName);
 Object value = beanWrapper.getValue(propertyName);
```

```
 System.out.println(type + ": " + value);
}
```

Sometimes you want to have control over bean instantiation and would like to set some "preferences" over how JuffrouBeanWrapper behaves. This is where the BeanWrapperContext comes in.

BeanWrapperContext

The BeanWrapperContext is the object that holds metadata for a BeanWrapper. This metadata is composed by data collected through class introspection, including the references to the getter and setter methods of the class and all classes it extends.

If you instantiate a BeanWrapper using the default constructor, it will create a new BeanWrapperContext. But if you instantiate a BeanWrapper by passing a BeanWrapperContext, no introspection overhead is needed.

Table 3.1. time in milliseconds to handle 10.000 BeanWrappers

Mode	Instantiation only	With property setting
Spring Framework's BeanWrapperImpl	64	291
JuffrouBeanWrapper	9	41
JuffrouBeanWrapper with BeanWrapperContext	1	20
JuffrouBeanWrapper with BeanWrapperFactory	9	20

JDK 6 build 35 running on a Windows 7 32bit (Intel i3 2,93GHz with 4GB ram) machine

When a JuffrouBeanWrapper creates a nested JuffrouBeanWrapper it also creates a nested BeanWrapperContext, of course. But it will only create one BeanWrapperContext per property type, so, for instance, if you have a BeanWrapper around a class Person with two properties (home and work) of type Address, you can have two nested JuffrouBeanWrappers (one for home and another for work), but only one nested BeanWrapperContext.

You can obtain the BeanWrapperContext of a JuffrouBeanWrapper at any time calling the method `beanWrapper.getContext()`.

Controling bean instantiation

With BeanWrapperContext you can define a class that will be called to instantiate the wrapped class as well as the classes of the nested beans whenever they need to be instantiated. To do this, first create a class that implements the interface `BeanInstanceBuilder`. See the following example:

```
BeanInstanceBuilder iCreator = new BeanInstanceBuilder() {
 @Override
 public Object build(Class clazz) throws BeanInstanceBuilderException {
  Programmer programmer = new Programmer();
  programmer.setLastName("Smith");
  return programmer;
 }
};
BeanWrapperContext context = BeanWrapperContext.create(Programmer.class);
context.setBeanInstanceBuilder(iCreator);
JuffrouBeanWrapper bw = new JuffrouBeanWrapper(context);
bw.setValue("firstName", "John");
Programmer programmer = (Programmer) bw.getBean();
Assert.assertEquals("John", programmer.getFirstName());
```

```
  Assert.assertEquals("Smith", programmer.getLastName());
```

You might also want to associate more information with a bean than the introspection information collected by the BeanWrapperContext. For instance you might want add information to help represent the bean in XML format (like Juffrou-XML does). This is where the CustomizableBeanWrapperFactory comes in.

CustomizableBeanWrapperFactory

The BeanWrapperFactory is the object responsible for creating all the BeanWrapperContexts. It does mainly three things:

- Keeps track of the BeanWrapperContexts it creates, so it doesn't create two BeanWrapperContexts for the same bean class.
- Injects itself into the BeanWrapperContexts it creates, so that they can use the same factory to create their nested BeanWrapperContexts.
- Gives out BeanWrapperContexts upon call to `factory.getBeanWrapperContext(Person.class)` method. The BeanWrapperContext will only be created in none exists for the specified class (in this case Person.class).

In fact you can also use the BeanWrapperFactory to instantiate JuffrouBeanWrappers using the methods `factory.getBeanWrapper(Person.class)` or `factory.getBeanWrapper(personIntance)` for example. This is as fast as instantiating a JuffrouBeanWrapper with a BeanWrapperContext parameter.

Extending the BeanWrapperContext

So if you want to extend the BeanWrapperContext and add to the bean metadata all you have to do is create a class that extends BeanWrapperContext and "tell" the CustomizableBeanWrapperFactory to instantiate your class instead of BeanWrapperContext.

Example 3.2. Class that extends BeanWrapperContext

```
public class MyBeanWrapperContext extends BeanWrapperContext {

 //TODO create properties to extend the context

 public MyBeanWrapperContext(CustomizableBeanWrapperFactory factory, Class clazz, Type... types) {
  super(factory, clazz, types);
  //TODO some initialization
 }
}
```

And how do we "tell" the CustomizableBeanWrapperFactory to use this as BeanWrapperContext? Easy. We create a class that implements *BeanContextBuilder* like this one:

```
public class MyContextBuilder implements BeanContextBuilder {

 @Override
 public MyBeanWrapperContext build(
      CustomizableBeanWrapperFactory factory, Class clazz, Type... types) {

  MyBeanWrapperContext context = new MyBeanWrapperContext(factory, clazz, types);
  return context;
 }
}
```

And then we "tell" CustomizableBeanWrapperFactory to use this builder. See the example bellow:

Example 3.3. Using a custom BeanWrapperContext

```
CustomizableBeanWrapperFactory factory = new CustomizableBeanWrapperFactory();
factory.setBeanContextBuilder(new MyContextBuilder());
JuffrouBeanWrapper myPersonWrapper = factory.getBeanWrapper(Person.class);
MyBeanWrapperContext context = (MyBeanWrapperContext) myPersonWrapper.getContext();
```

BeanConverter

The BeanConverter is a utility class to convert between two beans.

Given any two beans and a map that establishes which properties in bean 1 correspond to properties in bean 2, this class can be used to automatically obtain bean 1 from an instance of bean 2 and vice-versa.

ReflectionUtil

The ReflectionUtil is a utility class with several helper static methods. They are all well documented in Javadoc for an easy and direct reference.

Some of these methods are:

- `getMapFromBean` Transform a Java bean into a Map where the keys are the property names. If there are nested beans, then the key will be the path of property names in the form "prop1.prop2.prop2". Properties with null values are not put in the map.
- `getBeanFromMap` Fill up a java bean with the contents of a map where the keys are property names.

Chapter 4. Getting Started with Juffrou-XML

Introduction

Juffrou-XML is an open source java library to marshall beans to xml and back. The objective is to make this simple, logical and flexible.

With Juffrou-XML you have simplified marshalling, wich means that you can obtain the XML representation of complex structure of java beans with all its nested beans represented as nested elements, but you can also get a "flattened" XML representation with properties from the root bean and properties from the nested beans in a very simple manner.

Enough talk! Let's see how it works:

Installing

Maven projects

To start using Juffrou-XML in your maven project just add the following dependency:

```
<dependency>
 <groupId>net.sf.juffrou</groupId>
 <artifactId>juffrou-xml</artifactId>
 <version>2.1.8-SNAPSHOT</version>
</dependency>
```

This will allow you access the source code of the library as well as the javadoc files, if you have checked the options "download artifact sources" and "download atifact javadoc in your IDE."

Non maven projects

Download the file `juffrou-2.1.8-SNAPSHOT-bundle.zip` from the *website* [http://juffrou.sourceforge.net] and extract it's contents to a temporary directory.

Add `juffrou-reflect-2.1.8-SNAPSHOT.jar` and `juffrou-xml-2.1.8-SNAPSHOT.jar` to the classpath of your project and you are good to go.

With these libraries in your classpath you can start using juffrou-XML right away:

Example 4.1. Marshalling a java bean:

```
Person person = new Person();
person.setFirstName("Carlos");
person.setLastName("Martins");
person.setBirthDay(new SimpleDateFormat("yyyy-MM-dd").parse("1967-10-01"));

JuffrouXml juffrouXml = new JuffrouXml();
String xmlString = juffrouXml.toXml(person);
```

The output will be the following XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<net.sf.juffrou.xml.test.dom.Person>
 <firstName>Carlos</firstName>
 <lastName>Martins</lastName>
 <birthDay>10/1/67 12:00 AM</birthDay>
</net.sf.juffrou.xml.test.dom.Person>
```

Without configuration, the names of root elements will be the class name of the corresponding bean, all its properties will be marshalled and the names of the child elements will be the names of the corresponding property. Juffrou-XML is also able to unmarshall the XML text back to a person bean as long as the classes in the root elements are in the program classpath like in the following example:

Example 4.2. Unmarshalling from XML to Person:

```
Person person = (Person) juffrouXml.fromXml(xmlString);
```

Configuring Juffrou-XML

If you want to change the element names of the beans or how those beans are marshalled / unmarshalled, then you need configuration. Configuration can be done either through direct coding or by means of an XML file.

To configure by file you can instantiate Juffrou-XML and pass the file name in the constructor, or you can instantiate Juffrou-XML with the default constructor and then call the readConfigFile method.

Example 4.3. Configuring Juffrou-XML:

```
JuffrouXml juffrouXml = new JuffrouXml("classpath:juffrou-config.xml");

JuffrouXml juffrouXml = new JuffrouXml();
juffrouXml.readConfigFile("classpath:config-file-one.xml");
juffrouXml.readConfigFile("file:/etc/config-file-two.xml");
```

Defining root element names

Example 4.4. define root element names through code

```
JuffrouXml juffrouXml = new JuffrouXml();
juffrouXml.registerRootElement(Person.class, "Person");
```

Example 4.5. Defining root element in configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://juffrou.sourceforge.net/juffrou-xml"
 xsi:schemaLocation="http://juffrou.sourceforge.net/juffrou-xml
 http://juffrou.sourceforge.net/juffrou-xml/schemas/juffrou-xml.xsd">

 <root-element xml="Person" type="net.sf.juffrou.xml.test.dom.Person" />

</mapping>
```

The corresponding XML now looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Person>
 <firstName>Carlos</firstName>
```

```
 <lastName>Martins</lastName>
 <birthDay>10/1/67 12:00 AM</birthDay>
</Person>
```

Defining element names

Example 4.6. define element names through code

```
 JuffrouXml juffrouXml = new JuffrouXml();
 juffrouXml.registerElement(Person.class, "lastName", "Surname");
```

Example 4.7. Defining element names in configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://juffrou.sourceforge.net/juffrou-xml"
 xsi:schemaLocation="http://juffrou.sourceforge.net/juffrou-xml
 http://juffrou.sourceforge.net/juffrou-xml/schemas/juffrou-xml.xsd">

 <root-element xml="Person" type="net.sf.juffrou.xml.test.dom.Person">
  <element property="firstName" />
  <element property="lastName" xml="Surname" />
  <element property="birthDay" />
 </root-element>

</mapping>
```

The corresponding XML now looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Person>
 <firstName>Carlos</firstName>
 <Surname>Martins</Surname>
 <birthDay>10/1/67 12:00 AM</birthDay>
</Person>
```

Defining attributes

Example 4.8. define attributes through code

```
 JuffrouXml juffrouXml = new JuffrouXml();
 juffrouXml.registerAttribute(Person.class, "firstName", "name");
```

Example 4.9. Defining attributes in configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://juffrou.sourceforge.net/juffrou-xml"
 xsi:schemaLocation="http://juffrou.sourceforge.net/juffrou-xml
 http://juffrou.sourceforge.net/juffrou-xml/schemas/juffrou-xml.xsd">

 <root-element xml="Person" type="net.sf.juffrou.xml.test.dom.Person">
  <attribute property="firstName" xml="name" />
  <element property="lastName" xml="Surname" />
  <element property="birthDay" />
 </root-element>

</mapping>
```

The corresponding XML now looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Person name="Carlos">
 <Surname>Martins</Surname>
 <birthDay>10/1/67 12:00 AM</birthDay>
</Person>
```

Chapter 5. More Advanced Stuff

Serializers

Serializers are the classes responsible for translating bean property values to XML and back. Juffrou-xml comes with serializers for the basic java types, like `String`, `Integer` and `Boolean` for instance. *You can see the complete list of serializers in the javadoc for the package net.sf.juffrou.xml.serializer*.

You may want to create your own serializers and tell juffrou to use them. For intance, you may want to have properties of type `Date` displayed in a particular format, so you create a serailizer that knows how to convert between `Date` and that specific format. You can also use serializers to convert between the text in an XML element and a more complex type. For example, in a class called `Person`, with a property `private Address home` you may want to represent `home` as a single text string. In this case you create a serializer that knows how to convert between `Address` and that text string.

Creating your own serializer is simple. Just implement the **Serializer** interface like in the following example:

```
public class SimpleDateSerializer implements Serializer {

 private final SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

 @Override
 public void serialize(JuffrouWriter writer, BeanWrapper valueOwner,
   String valuePropertyName) {
  writer.write(formatter.format((Date)valueOwner.getValue(valuePropertyName)));
 }

 @Override
 public void deserialize(JuffrouReader reader, BeanWrapper valueOwner,
   String valuePropertyName) {
  String value = reader.getText();
  try {
   valueOwner.setValue(valuePropertyName, formatter.parse(value));
  } catch (ParseException e) {
  }
 }
}
```

Once your serializer class is created, you can use it to translate your beans or bean properties using by configuring through code or mapping file.

Example 5.1. using serializers through code

```
 JuffrouXml juffrouXml = new JuffrouXml();
 juffrouXml.registerSerializer("mySimpleDateSerializer", new SimpleDateSerializer());
 juffrouXml.registerElement(Person.class, "birthDay", "birthday", "mySimpleDateSerializer");
```

Example 5.2. using serializers through configuration file

When using a configuration file you can define reusable serializers to share between bean properties or define a serializer that is exlusive to a perticular bean property.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://juffrou.sourceforge.net/juffrou-xml"
 xsi:schemaLocation="http://juffrou.sourceforge.net/juffrou-xml
 http://juffrou.sourceforge.net/juffrou-xml/schemas/juffrou-xml.xsd">
```

```
 <serializer id="mySimpleDateSerializer"
  class="net.sf.juffrou.xml.test.dom.SimpleDateSerializer"/>

 <root-element xml="Person" type="net.sf.juffrou.xml.test.dom.Person">
  <attribute property="firstName" xml="name" />
  <element property="lastName" xml="Surname">
   <serializer class="net.sf.juffrou.xml.serializer.StringSerializer"/>
  </element>
  <element property="birthDay" xml="birthday">
   <serializer ref="mySimpleDateSerializer"/>
  </element>
 </root-element>

</mapping>
```

You can define serializers to share between bean properties with `serializer` element in the mapping file:

```
<serializer id="mySimpleDateSerializer" ❶
 class="net.sf.juffrou.xml.test.dom.SimpleDateSerializer"/>  ❷
```

❶ Id of the serializer to be referenced by the bean property serializers.
❷ Class name of the serializer. This class will be instantiated only once and will be shared between the bean properties that reference it.

You can define a serializer for a bean property by nesting a `serializer` element like in the above example. Below are all the properties for a property serializer:

```
<serializer
 class="net.sf.juffrou.xml.test.dom.SimpleDateSerializer"  ❶
 ref="mySimpleDateSerializer"  ❷
 bean="mySpringBeanSerializer" />  ❸
```

❶ Class name of the specific serializer class to use. This class will be instantiated and used exclusively for the bean property.
❷ Id of the shared serializer.
❸ Id of a springframework bean which implements `serializer`. This is only valid when using the library `juffrou-xml-spring`

Simplified Marshalling

Simplified marshalling is the possibility of marshalling nested beans into a "flat" XML structure like this: imagine that you don't want the whole home address marshalled for this person. You only want his home city and you want it displayed as it were a simple property of person.

Easy! All you have to do is this:

```
JuffrouXml juffrouXml = new JuffrouXml();
juffrouXml.registerRootElement(Person.class, "Person");
juffrouXml.registerElement(Person.class, "home.city", "homeTown", null);
String xmlString = juffrouXml.toXml(person);
```

The output will now be:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Person>
  <firstName>Carlos</firstName>
  <lastName>Martins</lastName>
```

```
  <birthDay>1967-10-01</birthDay>
  <homeTown>Lisboa</homeTown>
</Person>
```

Of course this XML will also be unmarshalled back to a person object. That person object will have a home with city Lisboa.

Chapter 6. Getting Started with Juffrou-XML-Spring

Introduction

Juffrou-XML-Spring is the implementation of springframework's (spring-oxm) Marshaller and Unmarshaller interfaces using Juffrou-XML.

Installing

Maven projects

To start using Juffrou-XML in your maven project just add the following dependency:

```
<dependency>
 <groupId>net.sf.juffrou</groupId>
 <artifactId>juffrou-xml-spring</artifactId>
 <version>2.1.8-SNAPSHOT</version>
</dependency>
```

This will allow you access the source code of the library as well as the javadoc files, if you have checked the options "download artifact sources" and "download atifact javadoc in your IDE."

Non maven projects

Download the file `juffrou-2.1.8-SNAPSHOT-bundle.zip` from the *website* [http://juffrou.sourceforge.net] and extract it's contents to a temporary directory.

Add `juffrou-reflect-2.1.8-SNAPSHOT.jar`, `juffrou-xml-2.1.8-SNAPSHOT.jar` and `juffrou-xml-spring-2.1.8-SNAPSHOT.jar` to the classpath of your project and you are good to go.

Configuring Juffrou-XML though Spring

In a spring application context, Juffrou-XML is a bean and can be configured like any other bean in spring.

Example 6.1. Juffrou-XML Spring Configuration

```
<bean id="marshaller" class="net.sf.juffrou.xml.JuffrouXmlSpring">
 <property name="mappingLocations">
  <list>
   <value>classpath:/net/sf/juffrou/**/*-xml-mapping.xml</value>
   <value>file:${CONFIG_LOCATION}/juffrou-xml/**/*-xml-mapping.xml</value>
  </list>
 </property>
</bean>
```

In the above example spring tells juffrou-xml to load its configuration from several mapping files.

Defining serializers as spring beans

Serializers may be used to convert between the text in an XML element and a complex type. With juffrou-xml-spring you can define your serializers as java beans and thus take advantage of spring's dependency injection mechanism.

For example purposes lets consider the `SimpleDateSerializer` from the spring-xml reference. First we would define the serializer bean:

```
<bean id="mySimpleDateSerializerBean" class="net.sf.juffrou.xml.test.dom.SimpleDateSerializer">

   <bean id="marshaller" class="net.sf.juffrou.xml.JuffrouMarshaller">
    <property name="mappingLocations">
     <list>
      <value>classpath:/net/sf/juffrou/**/*-xml-mapping.xml</value>
      <value>file:${CONFIG_LOCATION}/juffrou-xml/**/*-xml-mapping.xml</value>
     </list>
    </property>
   </bean>
```

To use this serializer bean in juffrou-xml you would configure the configuration mapping like the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://juffrou.sourceforge.net/juffrou-xml"
 xsi:schemaLocation="http://juffrou.sourceforge.net/juffrou-xml
 http://juffrou.sourceforge.net/juffrou-xml/schemas/juffrou-xml.xsd">

 <root-element xml="Person" type="net.sf.juffrou.xml.test.dom.Person">
  <attribute property="firstName" xml="name" />
  <element property="lastName" xml="Surname" />
  <element property="birthDay" xml="birthday">
   <serializer bean="mySimpleDateSerializerBean"/>
  </element>
 </root-element>

</mapping>
```