# JAJB-1.1.0 XML Configuration Documentation

JAJB configuration needs minimum two files:

The first, a file in witch you can describe all your converters and include in JAJB configuration all your mapping files.

The second (or second, third, forth,……etc.) is a file in witch you describe all your JavaBeans you need to mapping.

Let's start to look at first file:

In this example we'll use jajb-config.xml like name, but you can use the name you prefer.
First of all the file must start with version end encoding string like below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Then you must declare the doctype and the dtd like below:

```
<!DOCTYPE jajb-configure PUBLIC "//jajb/jajb-configure-1.0.dtd"
"http://jajb.sourceforge.net/dtds/jajb-configure-1.0.dtd" >
```

After the headline starts the file content: the content must be enclosed in jajb-configure tag like below:

```
<jajb-configure>
      ……………
      ……………
      ……………
</jajb-configure>
```

The content enclosed in jajb-config tag concerns the converters description end the JavaBean mapping files you want to include in JAJB configuration:

1) Converters description:

   Each converter you want describe must be enclosed in converters tag like below:

```
<jajb-configure>
      <converters>
            <converter id="myCoverter" type="my.package.MyConverter"/>
            <converter id="dateConverter" type="my.package.DateConverter"/>
            <converter id="intConverter" type="util.example.IntConverter"/>
      </converters>
</jajb-configure>
```

   Remember that for each converter, both id and type attribute are mandatory: you can use an unique name for single id attribute and the fully qualified name of your converter for to type attribute you want associate to chosen id. Remember that each your converter must implement `it.dangelo.javabinding.converters.Converter`.

2) Mapping files inclusion:

Each mapping file you want include in JAJB configuration must be enclosed in mappings tag like below:

```
<jajb-configure>
      <converters>
            <converter id="myCoverter" type="my.package.MyConverter"/>
            <converter id="dateConverter" type="my.package.DateConverter"/>
            <converter id="intConverter" type="util.example.IntConverter"/>
      </converters>
      <mappings>
            <mapping path="mapping1.xml" />
            <mapping path="../mapping2.xml" />
            <mapping path="test/mapping/beanMappingExample.xml" />
      </mappings>
</jajb-configure>
```

Obviously the unique path attribute is mandatory and must identify the path of correspondent mapping file relatively to the ClassLoader.

3) Proxy resolver description:

If you use jajb into a spring or cglib context you can describe a proxy resolver to resolve the real class type of your java bean, like below (remember that your proxy resolver must implement it.negro.jajb.proxies.ProxyResolver Interface):

```
<jajb-configure>
      <converters>
            <converter id="myCoverter" type="my.package.MyConverter"/>
            <converter id="dateConverter" type="my.package.DateConverter"/>
            <converter id="intConverter" type="util.example.IntConverter"/>
      </converters>
      <mappings>
            <mapping path="mapping1.xml" />
            <mapping path="../mapping2.xml" />
            <mapping path="test/mapping/beanMappingExample.xml" />
      </mappings>
      <proxy-resolver type="com.example.MyProxyResolver" />
</jajb-configure>
```

In jajb there are two implementation of ProxyResolver: it.negro.jajb.proxies.CGLIBProxyResolver for cglib proxies and it.negro.jajb.proxies.SpringJavaProxyResolver for spring or java proxies. If you need you can use them.

Now we can look at a mapping file example.

In this example we'll use mapping1.xml like name, but you can use the name you prefer, but respecting what we have wrote in jajb-config.xml concerning this mapping file.

First of all the file must start with version end encoding string like below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Then you must declare the doctype and the dtd like below:

```
<!DOCTYPE jajb-mappings PUBLIC "//jajb/mappings-1.0.dtd"
"http://jajb.sourceforge.net/dtds/mappings-1.0.dtd" >
```

After the headline starts the file content: the content must be enclosed in jajb-mappings tag like below:

```
<jajb-mappings>
      ...............
      ...............
      ...............
</jajb-mappings>
```

Now let's look at a single JavaBean mapping that must be enclosed in entity tag like below:

```
<jajb-mappings>
      <entity id="person" type="it.test.bean.Person" extends="animal">
            <fields>
                  <field name="name" type="java.lang.String"/>
                  <field name="surname" type="java.lang.String"/>
                  <field name="age" type="java.lang.Integer"/>
                  <field name="studies" type="java.lang.String" collectionType="array" />
                  <field name="divingLicense" factory="it.test.DrivingLicenseFactory"/>
                  <field name="birthDay" type="java.util.Date" converter="dateConverter"/>
                  <field name="el" property="elements" type="int" collectionType="array"/>
            </fields>
      </entity>
</jajb-mappings>
```

Concerning entity tag:

- **id**: you can initialize id attribute with your preferred name for bean. Id attribute is mandatory;
- **type**: you must initialize type attribute with fully qualified name of bean you want associate with chosen id. This attribute is mandatory.
- **extends**: if your bean is a class daughter of another class you must describe the mother class and initialize the extends attribute with the id chosen for the mother class, otherwise the inherited attribute of mother class don't will correctly managed.

Now let's talk about fields and field tag:
Each attribute of your bean (entity) must be declared using field tag, and all attributes of your entity must be enclosed in fields tag, like in the example.
The field tag, offers many attributes to correctly describe each entity attributes.
For each field you can/must set:

- **name**: represents the attribute name in the JSON representation of the entity; if you don't use attribute property, means that the bean attribute name is the same respect to JSON attribute name, so it must be equals to correspondent JSON attribute name and it can be not equals to correspondent bean attribute name. This attribute is mandatory.

- **property**: represents the bean attribute name you want to associate to JSON attribute name chosen for name attribute initialization. If JSON attribute name and bean attribute name are exactly equals you don't need to use property attribute: you can specify both equals JSON attribute name and bean attribute name using name attribute. This attribute isn't mandatory.
- **factory**, **refId**, **type**: these three attributes are globally described because there's a relation between their; first of all let's talk about attributes priority: the usage priority is factory-refId-type so if you use all three attributes, only factory will be considered for binding; if you use refId, converter and type, only refId will be considered for binding. That means if you want use one of these attributes you can don't use the other two, only one of these four attribute is mandatory according to your needs. Now look at specific attribute meaning:
    - **factory**: you must initialize this attribute with fully qualified name of your `it.dangelo.javabinding.factory.BindingFactory` interface implementation (See javabinding documentation for more info).
    - **refId**: if a field is another described bean (like drivingLicence in the example) you must initialize this attribute with chosen id for bean represented by this field.
    - **type**: you can initialize this attribute with fully qualified name of the field. Usually, this attribute is used to describe primitive or Wrappers type. If you use also collectionType attribute (see below) you must initialize this attribute with fully qualified name of components of collection.
- **collectionType**: you can use this attribute when the correspondent java bean attribute is an array or a collection. If you use this attribute the type attribute will be the component type of the array or collection. You can initialize this attribute with your collection class type or simply writing "array" if you need an array. This attribute isn't mandatory.
- **converter**: you should initialize this attribute with the chosen converter id you should have mapped in the configuration file (jajb-config in our example). If you use this attribute in couple with collectionType attribute, the collection's elements will be passed to the specified converter; if you want to pass the entire collection to the specified converter, don't use the collectionType attribute, but only converter attribute (and obviously name, property, refId, etc.). This attribute isn't mandatory.
- **setter**: you can use this attribute when in your JSONObject there's an attribute that corresponds to a private attribute in your entity. You can initialize this attribute with %number (for example %2) that represents its index like parameter of the constructor that will be invoked build your java bean. In this case you have to implement a constructor for non accessible attribute with correct type and order of parameters; otherwise, instead of %number, you can initialize this attribute with the setter name of the field you' re describing if it doesn't respect the java beans naming convention. Remember that if you define a particular constructor, you should explicitly redefine de default constructor! This attribute isn't mandatory.
- **getter**: you can use this attribute when in your JavaBean the getter correspondent to field in witch you are using it doesn't respect the naming convention (for example: if your java bean attribute name is dog, but its getter name isn't getDog but retrieveDog you can initialize the attribute getter with "retreiveDog" string. This attribute isn't mandatory.

In the following example all JAJB features are used. There’s the JavaBeans implementation in java language at the first and then their mappings (remember you can initialize the refId attribute with ids existing also in other files respect to the file in witch you are using it).

```
//Class Person
package bean;

import java.util.Date;


public class Person {
	private String name;
	private String surname;
	private Integer age;
	private String[] studyLevels;
	private DrivingLicence drivingLicence;
	private Date birthday;
	private int[] elements;
	private List<String> moreElements;
	private Dog dog;

	public Person() {}
	//LOOK AT THE CONSTRUCOTR FOR ATTRIBUTE age that have setter="%0" in configuration file
	public Person(Integer age) {
		this.age = age;
	}

	public Dog getDog() {
		return dog;
	}
	public void setDog(Dog dog) {
		this.dog = dog;
	}
	public List<String> getMoreElements() {
		return moreElements;
	}
	public void setMoreElements(List<String> moreElements) {
		this.moreElements = moreElements;
	}
	public int[] getElements() {
		return elements;
	}
	public void setElements(int[] elements) {
		this.elements = elements;
	}
	public Date getBirthDate() {
		return birthDate;
	}
	public void setBirthDay(Date birthDay) {
		this.birthDate = birthDay;
	}
	public String getSurname() {
		return surname;
	}
	public void setSurname(String surname) {
		this.surname = surname;
	}
	public Integer getAge() {
		return age;
	}
	public String retieveNameValue() {
		return name;
	}
	public void setNameValue(String name) {
```

```
            this.name = name;
      }
      public DrivingLicence getDrivingLicence() {
            return drivingLicence;
      }
      public void setDrivingLicence(DrivingLicence drivingLicence) {
            this.drivingLicence = drivingLicence;
      }
      public String[] getStudyLevels() {
            return studyLevels;
      }
      public void setStudyLevels(String[] studyLevels) {
            this.studyLevels= studyLevels;
      }
}


//Class DrivingLicence
package bean;

public class DrivingLicence {
      private String nation;
      private Integer year;
      public Integer getYear() {
            return year;
      }
      public void setYear(Integer year) {
            this.year= year;
      }
      public String getNation() {
            return nation;
      }
      public void setNation(String nation) {
            this.nation= nation;
      }
}


//Class DrivingLicenceSpecial
package bean;

public class DrivingLicenceSpecial extends DrivingLicence {
      private String hp;
      public String getHp() {
            return hp;
      }
      public void setHp (String hp) {
            this.hp = hp;
      }
}


//Class Dog
package bean;

public class Dog {
      private String race;
      public String getRace() {
            return race;
      }
      public void setRace (String race) {
            this.race = race;
      }
}
```

Now let's look at the mapping file of this javabeans structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jajb-mappings PUBLIC "//jajb/mappings-1.0.dtd"
"http://jajb.sourceforge.net/dtds/mappings-1.0.dtd" >
<jajb-mappings>

      <entity id="$person" type="bean.Persona" >
            <fields>
                  <field name="name" type="java.lang.String" setter="setNameValue"
                         getter="retreveNameValue"/>
                  <field name="surname" type="java.lang.String"/>
                  <field name="age" type="java.lang.Integer" setter="%0"/>
                  <field name="studyLevels" type="java.lang.String" collectionType="array" />
                  <field name="drivingLicence" factory="binding.LicenseBindingFactory"/>
                  <field name="birthday" converter="dateConverter"/>
                  <field name="dog" refId="dog"/>
                  <field name="el" property="elements" type="int" collectionType="array"/>
                  <field name="moreElements" type="java.lang.String"
                         collectionType="java.util.ArrayList"/>
            </fields>
      </entity>

      <entity id="$drivingLicence" type="bean.DrivingLicence">
            <fields>
                  <field name="nation" type="java.lang.String" />
                  <field name="year" type="java.lang.Integer" />
            </fields>
      </entity>

      <entity id="$drivingLicenceSpecial" type="bean.DrivingLicenceSpecial"
              extends="$drivingLicence">
            <fields>
                  <field name="hp" type="java.lang.String" />
            </fields>
      </entity>

      <entity id="$dog" type="bean.Dog">
            <fields>
                  <field name="race" type="java.lang.String" />
            </fields>
      </entity>

</jajb-mappings>
```

ENJOY!