JAJB-1.1.0 Annotation Configuration Documentation

What do you need?
Just the two JAJB annotation: @EntityAnnotation and @FieldAnnotation.

1) @EntityAnnotation: is a class annotation that you can apply to the bean you want to map; it has only one attribute (extends) not mandatory. The extends attribute says witch is the jajb id of the superclass; its value can be the id of a class described in xml context (for example: '$person'), or the id of another class described using annotation (for example: '@it.example.person') or the id of a class not described at all (for example: '#it.example.person').
   Remember that the jajb id of an annotated class stars always with '@' and continues with the full class name; the jajb id of a not described class starts always with '#' and continues with the full class name; the jajb id of a class described in the xml context starts always with '$' and continues with the choosen name.

   Now let's look to the following example:
   We have the class Person and the class European that is subclass of Person:

   1. you can describe both them in xml context like below:

```
<jajb-mappings>
      <entity id="$person" type="it.test.bean.Person">
            <fields>
                  <field name="name" type="java.lang.String"/>
                  ………………………………………………
                  ………………………………………………
            </fields>
      </entity>

      <entity id="$european" type="it.test.bean.European" extends="$person">
            <fields>
                  ………………………………………………
                  ………………………………………………
            </fields>
      </entity>


                  ………………………………………………
                  ………………………………………………

</jajb-mappings>
```

   2. you can describe Person in xml context and European using @EntityAnnotation:

```
<jajb-mappings>
      <entity id="$person" type="it.test.bean.Person" >
            <fields>
                  ………………………………………………
                  ………………………………………………
            </fields>
      </entity>

                  ………………………………………………


</jajb-mappings>
```

```
@EntityAnnotation(extend="$person")
public class European extends Person {

}
```

3. you can describe Person using @EntityAnnotation and European in xml context:

```
<jajb-mappings>
      <entity id="$european" type="it.test.bean.European"
      extends="@it.bean.test.Person">
            <fields>
                  ………………………………………………
                  ………………………………………………
            </fields>
      </entity>

                  ………………………………………………


</jajb-mappings>
```

```
@EntityAnnotation
public class Person {

}
```

4. you can describe both Person and Eurpean using @EntityAnnotation:

```
@EntityAnnotation
public class Person {

}

@EntityAnnotation(extend="@it.bean.test.Person")
public class European extends Person {

}
```

5. you can use also classes not described at all using like jajb id a string that starts with ‘#’ and continues with the class name, for example fore the class person ‘#it.bean.test.Person’; you can make a complete combination between xml context, annotation context and no description; you can also describe the same class in both xml and annotation context and use at the same time the three possible kind of mapping by using desired jajb id.

2) @FieldAnnotation: is a method annotation that you can apply generally to the getter method of the bean you want to map; now let's look at the @FiledAnnotation attributes:

- **name**: represents the attribute name in the JSON representation of the entity; if you don't use property attribute, 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 the class of your `it.dangelo.javabinding.factory.BindingFactory` interface implementation (See javabinding documentation for more info).
  - **refId**: if a field is another described bean you must initialize this attribute with chosen id for bean represented by this field. For possible values of refId, look at the considerations made before about extend attribute of @EntityAnnotation
  - **type**: you can initialize this attribute with the Class 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 the Class 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 or array class type. This attribute isn't mandatory.
- **converter**: you should initialize this attribute with the class of your particular implementation of `it.dangelo.javabinding.converters.Converter`. 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 the 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.

Now let's look to the following complete example:

```
//Class Person
package bean;

import java.util.Date;

@EntityAnnotation
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 its own
@FieldAnnotation
      public Person(Integer age) {
            this.age = age;
      }



@FieldAnnotation(name="dog", refId="#bean.Dog")
      public Dog getDog() {
            return dog;
      }
      public void setDog(Dog dog) {
            this.dog = dog;
      }



@FieldAnnotation(name="moreElements", type=String.class ,collectionType=ArrayList.class)
      public List<String> getMoreElements() {
            return moreElements;
      }
      public void setMoreElements(List<String> moreElements) {
            this.moreElements = moreElements;
      }



@FieldAnnotation(name="elements", type=Integer.TYPE ,collectionType=int[].class)
      public int[] getElements() {
            return elements;
      }
      public void setElements(int[] elements) {
            this.elements = elements;
      }
```

```java
@FieldAnnotation(name="birthDate",collectionType=Date[].class,converter=DateConverter.class)
          public Date getBirthDate() {
                return birthDate;
          }
          public void setBirthDay(Date birthDay) {
                this.birthDate = birthDay;
          }



  @FieldAnnotation(name="surname", type=String.class)
          public String getSurname() {
                return surname;
          }
          public void setSurname(String surname) {
                this.surname = surname;
          }



  @FieldAnnotation(name="age", type=Integer.class, setter="%0")
          public Integer getAge() {
                return age;
          }



  @FieldAnnotation(name="nome", type=String.class, getter="retrieveNameValue")
          public String retrieveNameValue() {
                return name;
          }
          public void setNameValue(String name) {
                this.name = name;
          }



  @FieldAnnotation(name="drivingLicence", factory=LicenceBindingFactory.class)
          public DrivingLicence getDrivingLicence() {
                return drivingLicence;
          }
          public void setDrivingLicence(DrivingLicence drivingLicence) {
                this.drivingLicence = drivingLicence;
          }



  @FieldAnnotation(name="studyLevels", type=String.class  collectionType=String[].class)
          public String[] getStudyLevels() {
                return studyLevels;
          }
          public void setStudyLevels(String[] studyLevels) {
                this.studyLevels= studyLevels;
          }
    }
```

```java
//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 talk about refId attribute for the last time:

For the dog property we have defined refId="#bean.Dog". That means jajb will make a direct correspondence between Dog class and dog JSONObject; but we can use also use "@bean.Dog" (annotated refId) that means jajb will use the annotation configuration information for Dog class; obviously, if Dog hasn't @EntityAnnotation an exception will be threw; at least we can use "$mydog" (xml context refId) that means jajb will use the xml context configuration information for Dog class; obviously if you didn't describe Dog into xml context or "$mydog" is a wrong refId an exception will be threw.

At least, how to instantiate JAJB?

If you use an xml context you should instantiate JAJB like below:

```
InputStream xmlContext = new FileInputStream("/home/gabriele/jajb-config.xml");
JAJB jajb = JAJB.newInstance(xmlContext);
```

Else if you don't use an xml context you can instantiate JAJB like below:

```
JAJB jajb = JAJB.newInstance();
```

Remember you must always apply a cast to unmarshalling methods result like below:

We suppose to have a jsonString that represents a Person object and we want to unmarshall this jsonString using the annotated description of Person class:

```
String personJson = "{\"name\":\"gabriele\",\"surname\":\"negro\", ....... and so on}";
Person person = (Person)jajb.unmarshall(personJsonString, "@bean.Person");
```

ENJOY!