Getting Started

Here's a simple example of using JiBX. Suppose you have the following XML customer information document structure:

```
<customer>
  <person>
    <cust-num>123456789</cust-num>
    <first-name>John</first-name>
    <last-name>Smith</last-name>
  </person>
  <street>12345 Happy Lane</street>
  <city>Plunk</city>
  <state>WA</state>
  <zip>98059</zip>
  <phone>888.555.1234</phone>
</customer>
```

JiBX provides great flexibility in binding your XML document structure to Java objects, so there are really many different ways you can represent this data using JiBX. For right now I'll stay with a Java object model that matches the XML document structure, and I'll just use plain Java data classes (fields only) to keep everything compact. Here's what these might look like for my customer document:

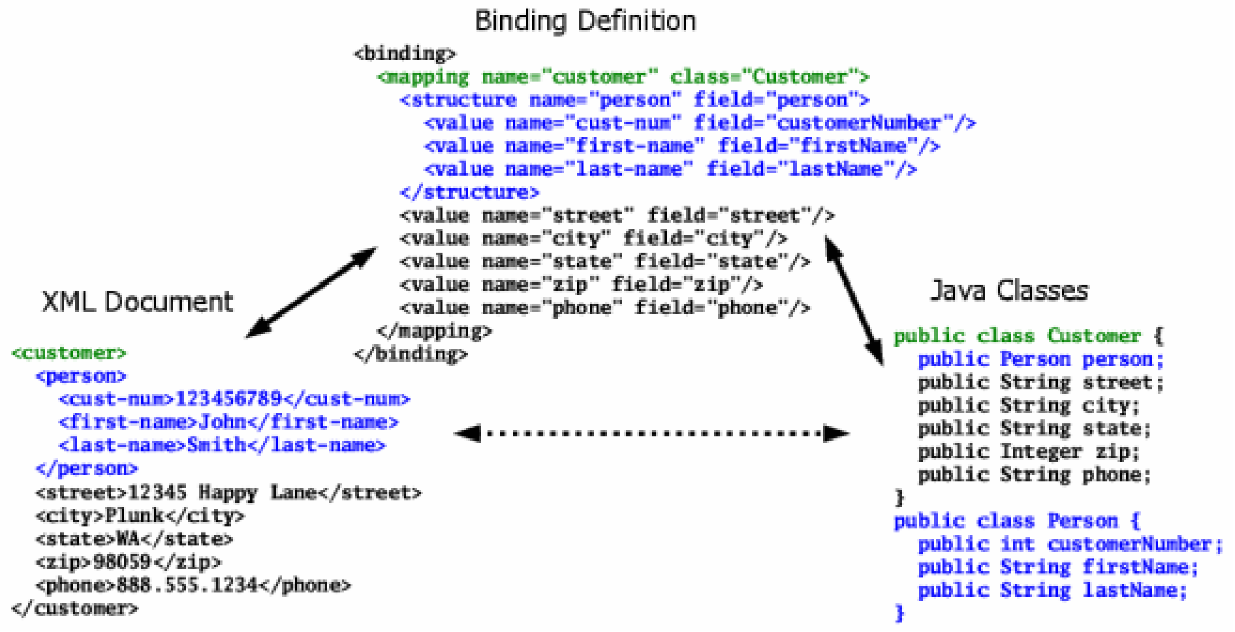
```
  public class Customer {
    public Person person;
    public String street;
    public String city;
    public String state;
    public Integer zip;
    public String phone;
  }

  public class Person {
    public int customerNumber;
    public String firstName;
    public String lastName;
  }
```

This is keeping almost all the data as `String`s, with the exception of the customer number and the zip code. These are both values with natural representations as numbers, so I've gone ahead and expressed them that way in the Java classes. I used an `int` for the customer number because I can be sure I've got one of those. For the zip code I've instead used an `Integer`, as an example of a simple object type.

Figure 1 shows the XML document, the Java classes, and a JiBX binding that connects the two. I've highlighted in green the top-level connection between the **customer** element and the `Customer` class, linked by the **mapping** element in the binding definition; and in blue the connection between the **person** element and the `Person` class, linked by the **structure** element in the binding definition.

Figure 1. Simple binding example



```
<binding>
  <mapping name="customer" class="Customer">
    <structure name="person" field="person">
      <value name="cust-num" field="customerNumber"/>
      <value name="first-name" field="firstName"/>
      <value name="last-name" field="lastName"/>
    </structure>
    <value name="street" field="street"/>
    <value name="city" field="city"/>
    <value name="state" field="state"/>
    <value name="zip" field="zip"/>
    <value name="phone" field="phone"/>
  </mapping>
</binding>
```

```
<customer>
  <person>
    <cust-num>123456789</cust-num>
    <first-name>John</first-name>
    <last-name>Smith</last-name>
  </person>
  <street>12345 Happy Lane</street>
  <city>Plunk</city>
  <state>WA</state>
  <zip>98059</zip>
  <phone>888.555.1234</phone>
</customer>
```

```
public class Customer {
  public Person person;
  public String street;
  public String city;
  public String state;
  public Integer zip;
  public String phone;
}
public class Person {
  public int customerNumber;
  public String firstName;
  public String lastName;
}
```

Digging into the binding

The **mapping** element in the Figure 1 binding definition relates the named element (in this case **customer**) to a particular class (`Customer`). JiBX uses the defined mapping as a default for both marshalling instances of the class and unmarshalling occurrences of the element. Mappings can be nested, in which case the inner mappings are only active while marshalling or unmarshalling with the outer mapping. ***Global*** mappings - ones which are *not* nested inside other mappings - define elements that can be root elements of the XML document, and classes that can be root objects of the object structure. In Figure 1, the mapping from element **customer** to Java class `Customer` is a global mapping.

The **structure** element in the binding definition defines the handling of an element or object class within a particular context. In this case the context of the **structure** element is the **mapping** from element **customer** to class `Customer`. The **structure** element is a very versatile player in JiBX binding definitions, with variations used for several purposes. In the Figure 1 binding it's playing its basic role, with both an element name and an object reference (to the `person` field of the `Customer` class) supplied. This works very similarly to a **mapping** definition, but is specific to the context rather than setting a general rule.

This can be a confusing issue. Since **mapping** and **structure** have a lot in common, why use one instead of the other? Well, there are cases where you *have* to use one or the other. The root element for a document to be unmarshalled must have a **mapping**, as must the class of a root object to be marshalled. On the other hand, there are things you can do with a **structure** element that you can't do with a **mapping** (some of which you'll see in a later section of this tutorial, Structure mapping). A good principle to start with is to use a **mapping** only for the root element of your document (or the class of your root object, whichever way you prefer to see this). Later in this tutorial you'll learn about other circumstances where a **mapping** should be used, but only using it for the root element/class works fine for now.

JiBX bindings include a number of elements with attributes that reference classes, such as the **mapping** element in the Figure 1 binding. In this tutorial I've kept the bindings as simple as possible by using the default package for all the sample classes. When you're using JiBX with your own classes you'll need to remember to use *fully-qualified* names for all your classes (with leading package information, such as `org.jibx.runtime.Utility`).

Just to finish up with the Figure 1 binding, besides the **mapping** and **structure** elements I've already discussed, all the other components of the binding definition are **value** elements. These are the grunt workers of the binding definition, handling a single text component. In XML terms, the text component may be an attribute, element, ordinary character data, or CDATA section. On the Java side, the text component may be a value of any primitive or object type that has a defined conversion to and from a simple `String` value (JiBX has a set of built-in basic conversions, and you can easily define extensions using static serialize and deserialize methods). In the Figure 1 binding, all the **value** elements are defining bindings between an XML element name and a corresponding Java class field.

Next: Binding Extras