<structure> Element Definition

The **structure** element defines a structure component of a binding. This can take any of several forms. The first is where the structure is a complex XML element (one with attributes or child elements) that's linked to an object property of the containing object type. This is the most common form of usage. It's essentially equivalent to an "in-line" mapping definition. This variation applies when both an element name and an object property are defined by the **structure** element (or when the property is implied rather than defined directly, as when the structure element is the child of a collection element).

Variations of the **structure** element that define either an element name or a linked property value (but not both) are used for **structure mapping**. Each of these variations works differently depending on whether the **structure** element is empty (does not have any child elements). The full set of structure mapping variations is shown in the table below.

Structure Mapping Variations

Defines	Empty?	What it Does
Element	yes	Adds an empty element to the XML when marshalling (unless optional, in which case it's ignored when marshalling), discards and ignores the named element when unmarshalling.
	no	Structure mapping that defines a complex element (one with attributes or child elements) in the XML representation with no corresponding object. Child attribute or element values are in this case linked directly to properties of the containing object type.
Property	yes	References the mapping for the property object type, which must be defined within some enclosing context of this definition. If the property type exactly matches a defined mapping, that mapping will be used; if not, the mapping will be determined by the element

name when unmarshalling or the actual object type when marshalling. This allows generic `java.lang.Object` references to be used, as long as a mapping is defined for the actual type of the item. The only restriction is that if this type of reference is defined as optional it must be the last element in a list (otherwise, if the expected element is missing in a document being unmarshalled the next sibling element will be used instead).

no — Structure mapping that defines an object with no corresponding XML element. Attributes or child elements defined within this **structure** are part of the XML element defined by the containing structure or mapping. Optional structures with no associated element name are subject to some unmarshalling limitations. If the structure defines both attribute and element values, the presence or absence of the structure is determined based only on the attributes - if one of the defined attributes is present on the containing element the structure is considered to be present, otherwise it is not and the property will be set to `null`.

Structures with child definitions may be ordered (where the XML components bound to the child definitions occur in a particular sequence) or unordered (where the XML components can occur in any order), as determined by the **ordered** attribute of the structure attribute group. In the case of unordered structures only optional components are allowed as child definitions.

A **structure** element can be used without either an element name or a linked property value when you just want to say that some child elements make up an unordered set:

```
<mapping name="alphas" class="AlphabetBean">
  <value name="a" field="a"/>
  <structure ordered="false">
```

```
    <value name="b" field="b" usage="optional"/>
    <value name="c" field="c" usage="optional"/>
  </structure>
  <value name="d" field="d"/>
</mapping>
```

The above binding definition unmarshals both the element sequences (a, b, c, d) and (a, c, b, d). Since the unordered elements are (and have to be) optional, though, the binding will also allow (a, b, d), (a, c, d), and just (a, d). The unordered elements may also be duplicated without an error, so (a, c, b, c, d) would also be accepted (with repeated values stored in order, so that the final value for a property would be that of the last instance of the element). When marshalling an unordered group the sequence used in the binding definition will always be followed (so (a, c, b, d) would marshal as (a, b, c, d)). If you need to control the order of elements (or preserve the order, when starting from an unmarshalled document) you'll need to instead work with objects that can be held in a collection.

When a **structure** element is used with an object property JiBX will reuse an existing instance of the object during unmarshalling. The way this works is that JiBX only creates a new instance of the object for use in unmarshalling if the property value is `null`. If an element name is used with an optional structure, and that name is missing from an input XML document, the object property for that structure will always be set to `null` (since the representation is missing from the XML). If you don't want the property to ever be set to `null`, use a wrapper **structure** element for the optional element name around the actual structure definition.

The **structure** element supports one unique attribute along with several common attribute groups, listed below. The unique attribute is used to reference **mapping** definitions for objects. It may only be used when a property is supplied (or implied).

Attributes

map-as

This optional attribute can be used to override the type (or type name) to be used for a mapping reference. If used as a type, the value of this attribute must be the fully-qualified class name for the mapped type, and the specified class must be assignment compatible with the actual

property type (so you can't map a `java.lang.Integer` field as a `java.lang.String`, for instance, but *can* map a field typed `java.lang.Object` as a `java.lang.String`). If used as a type name, the name must match the type name defined by an abstract mapping. A **structure** element with no property reference can use the **map-as** attribute to invoke a specified mapping for the `this` object (where the mapping must be assignment compatible with the `this` object type). As of JiBX 1.1, the value of this attribute is interpreted as namespace qualified.

style

A **value-style** attribute present on the **structure** element sets a default for all contained elements. See the style attribute group description for usage details.

name

Attributes from the name group define an element mapped to the structure definition as a whole. The element defined in this way will be a wrapper for the XML representations of all child values defined by the structure. The name is optional unless this structure defines a mapping reference to the property type (property definition with no children, see the third row of the table at the top of this page), in which case it's forbidden. See the name attribute group description for usage details.

object

Attributes from the object group define the way object instances are created and used in marshalling and unmarshalling. See the object attribute group description for usage details.

property

Attributes from the property group define a property value, including how it is accessed and whether it is optional or required. See the property attribute group description for usage details.

structure

Attributes from the structure group define ordering and reuse of child binding components. See the structure attribute group description for usage details.