Attribute Groups

- style - controls simple value style (as attribute or element)
- name - defines XML attribute or element name
- object - provides options for Java object handling
- property - controls how a value is accessed from a Java object
- structure - define ordering and reuse of lists of child binding components
- string - define conversions between Java property values and text representations

All these attributes are optional unless stated otherwise.

style

value-style — Defines the binding style for simple values. The allowed values are "element" (values as child elements with only text content) or "attribute" (values as attributes).

name

name — Local (unqualified) name of element or attribute.

ns — Gives the namespace URI for the element or attribute name. If this is not used the default value is the innermost default namespace for this type, if any.

object

create-type — Gives the type to be used when creating instances of the object during unmarshalling. This gives an alternative to the **factory** attribute when all you want to do is use a specific implementation for an interface or an abstract class.

factory — Defines a factory method for constructing new instances of an object type. This applies to bindings for unmarshalling only, and if supplied it must be in the form of a fully-qualified class and method name (e.g.,

	"com.sosnoski.jibx.ObjectBuilderFactory.newInstance" specifies the newInstance() method of the ObjectBuilderFactory class in the com.sosnoski.jibx package) for a static method returning an instance of the bound class. As with the other methods in this group (**pre-set**, **post-set**, and **pre-get**), three different method signatures are allowed: No arguments; a single argument of type `java.lang.Object`, in which case the owning object is passed in the method call; or a single argument of type `org.jibx.runtime.IUnmarshallingContext`, in which case the unmarshalling context is passed in the method call (this allows access to the entire stack of objects being unmarshalled). If not supplied, instances of the bound class are constructed using a null argument constructor.
marshaller	Defines a custom serialization handler class, as the fully-qualified name of a class implementing the `org.jibx.runtime.Marshaller` interface. This is only allowed with an output binding; it is required if an unmarshaller is defined for an input-output binding.
nillable	Allows the W3C XML Schema attribute **xsi:nil="true"** to be used on an element in instance documents to indicate that the corresponding object is `null`. The default value is "false", set this attribute "true" to enable **xsi:nil** support. The marshalling behavior when the attribute is "true" and the object reference is `null` depends on whether the binding defines the corresponding element as optional or required. If the element is optional it will simply be left out of the marshalled document. If the element is required it will be written with an **xsi:nil="true"** attribute. This attribute can only be used with objects that are bound to an element name.
post-set	Defines a bound class method called on instances of the class after they are populated with data from unmarshalling. This can be used for any postprocessing

or validation required by the class. Three different method signatures are supported, as described in the **factory** attribute text.

pre-get

Defines a bound class method called on new instances of the class before they are marshalled. This can be used for any preprocessing or validation required by the class. Three different method signatures are supported, as described in the **factory** attribute text.

pre-set

Defines a bound class method called on new instances of the class before they are populated with data from unmarshalling. This can be used for any initialization or special handling required before a constructed instance is used. Three different method signatures are supported, as described in the **factory** attribute text.

unmarshaller

Defines a custom deserialization handler class, as the fully-qualified name of a class implementing the `org.jibx.runtime.Unmarshaller` interface. This attribute cannot be used in combination with the **factory**, or **pre-set** attributes. It is only allowed with an input binding; it is required if a marshaller is defined for an input-output binding.

property

field

Gives the name of the field within the containing class that supplies the property value. This is required except for auto-generated identity fields, for values from a collection, or when both **get-method** (for output bindings) and **set-method** (for input bindings) definitions are supplied.

flag-method

Defines a method to be called by JiBX to indicate the presence or absence of the associated element (or attribute, though this is mainly useful with elements). This is the name of a method taking a `boolean` parameter, which will be called with the value `true` if the element is present or the

value `false` if it is not present. This can be used in combination with the **test-method** attribute to implement a presence flag for an optional element which can be used even if the content of the element is ignored (by using an empty **structure** definition).

get-method

Defines a "get" method for retrieving the property value from an instance of the containing class. This is the name of a no-argument method returning a value (primitive or object). If a **get-method** is defined for an object value represented by some form of structure in the binding (not just a simple **value**, in other words), the method will be used to retrieve the current instance of an object when unmarshalling. This follows the principle of JiBX reusing existing objects for unmarshalled data where possible. If you return a `null` value during unmarshalling, JiBX will create a new instance of the object for unmarshalled data.

set-method

Defines a "set" method for storing the property value in an instance of the containing class. This is the name of a method with return type `void`, taking a single value (primitive or object) as a parameter. If both **get-method** and **set-method** are defined, the **set-method** parameter type must be the same as the **get-method** return value.

test-method

Defines a method for checking if an optional property is present in an instance of the containing class. This is the name of a no-argument method with return type `boolean`, which must return `true` if the property is present and `false` if it is not present. This is only allowed in combination with **usage="optional"**. If not specified, a simple `==` comparison is used with primitive types to check for a value different from the default, and a `equals()` comparison for object types with non-null defaults

type

Supplies the fully-qualified class name for the property value. This can be used to force a more specific type for a property value defined by the field definition or access method signature as either a base class or an interface.

usage — Defines the usage requirement for this property. The value can either be "required" (property is always present, the default if not specified) or "optional" (property is optional).

structure

allow-repeats — Determines whether repeated elements within an unordered group should be allowed. The default is "false", meaning that if a bound element is repeated the runtime code will throw an exception. Setting this "true" means repeated elements will be processed the same as in pre-1.1 versions of JiBX. A "true" value for this attribute is only allowed when all child definitions are elements (no attributes or text), and requires **ordered="false"**. It cannot be used in combination with **choice="true"**. This attribute is ignored on a **collection** element.

choice — Defines whether child binding definitions represent a choice between alternatives, with only one allowed (value "true") or a set of possibilities of which one or more may be present ("false", the default). A "true" value for this attribute is only allowed when all child definitions are elements (no attributes or text), and requires **ordered="false"**. It cannot be used in combination with **allow-repeats="true"** or **flexible="true"**. This attribute is ignored on a **collection** element.

flexible — Defines whether unknown elements within an unordered group should be ignored. The default is "false", meaning that if an unknown element (one not allowed by the binding) is found during unmarshalling the runtime code will throw an exception. Setting this "true" means unknown elements will be ignored (along with all their content). A "true" value for this attribute is only allowed when all child definitions are elements (no attributes or text), and requires **ordered="false"**. It cannot be used in combination with **choice="true"**. This attribute is ignored on a **collection** element.

label

Gives a label allowing the list of child components to be referenced from elsewhere in the binding definition. **Note that this technique has been deprecated, and will not be supported in JiBX 2.0. In most cases an abstract mapping can be used as a replacement.**

ordered

Defines whether child binding definitions represent an ordered list (value "true", the default) or an unordered set ("false"). When this is set "true", each child **value** component must define either an element or an attribute name (attributes are always unordered, so the **ordered** setting of the grouping has no effect on attributes). **value** elements defining text values (**style="text"**) are not allowed as direct children of groups with **ordered="false"**.

using

References a list of child components defined elsewhere in the binding definition. The value must match the **label** value used on a **mapping**, **structure**, or **collection** element somewhere in the binding definition. The child binding components of the referenced element are used as the content of the element making the reference. The object types associated with the binding definition element making the reference and that defining the reference must match, and the **order** established by the element that defined the reference determines whether the child definitions are considered as ordered or unordered. The element with this attribute must not have any child definitions. **Note that this technique has been deprecated, and will not be supported in JiBX 2.0. In most cases an abstract mapping can be used as a replacement.**

string

default

Gives the default value for a conversion. This is only allowed for optional properties. If not specified, the default for primitive types is the same as the member variable initial state defined by the JLS, and for object types is "null".

enum-

value-method	Specifies a method to be used to obtain the XML text representation for a Java 5 enum class. If specified, this value method is used for both marshalling and unmarshalling instances of the enum class (in the unmarshalling case, by checking each instance of the enum in turn until one is found matching the input text). If not specified, the `toString()` method of the enum class is instead used for marshalling, and the static `valueOf` enum method is used for unmarshalling, both of which use the enum value name directly.
deserializer	Defines a custom deserialization handler method, as the fully-qualified name of a static method with the signature `Target xxxx(String text)`, where `xxxx` is the method name and `Target` is the type of the property (primitive or object type). Note that when a custom deserialization handler method is used for an optional object type with no default value, that method will be called with a `null` argument when the corresponding value is missing in the input document. It's up to the handler method to handle this case appropriately (by returning either a `null` or an object of the expected type).
serializer	Defines a custom serialization handler method, as the fully-qualified name of a static method with the signature `String xxxx(Target value)`, where `xxxx` is the method name and `Target` is the type of the property (primitive or object type).
whitespace	Selects how whitespace will be handled when deserializing values. This optional attribute can have the values: "preserve", meaning all text is processed just as provided by the parser; "replace", meaning all tab, newline, and carriage return characters in the text are replaced by space characters before the text is processed; "collapse", meaning that after "replace" processing is done all leading and trailing space characters are eliminated and all embedded sequences of multiple spaces are replaced by single spaces; or

"trim", meaning that after "replace" processing is done all leading and trailing space characters are eliminated.