<mapping> Element Definition

The **mapping** element defines the binding used for an object type (instances of a particular class) within a context. A normal (or "concrete") mapping links an object type to a particular XML element. This linkage means that only one concrete mapping can be active (in context) for a particular Java class at a time, and the element names used for concrete mappings within a context must be unique. Only concrete mappings defined as children of the root **binding** element can represent the root object for a marshalling or the root element for an unmarshalling.

An abstract mapping defines a reusable XML representation for an object type, with the actual element name defined at the point of use. This makes abstract mappings roughly equivalent to schema named complex types. Unlike schema complex types, it's also possible to use abstract mappings without an element name (so that the mapping defines a grouping of attributes and or elements that are incorporated directly into the structure at the point of reference). Multiple abstract mappings can be defined for the same object type within a context, as long as different **type-name**s are used for the mappings.

Both abstract and normal mappings can be used as bases for extension mappings. Extension mappings provide an easy way of handling polymorphism in your Java code, using the equivalent of schema substitution groups. A base mapping can be extended by one or more other mappings using the **extends** attribute, and the extension mappings can then be used anywhere the base mapping is referenced within the binding. When marshalling, JiBX interprets the actual type of the object to find the appropriate extension mapping. When unmarshalling, JiBX uses the actual element name found to identify the extension mapping.

Although an abstract mapping can be used as the base for extension mappings, an abstract mapping can only extend another mapping if the abstract extension is in turn extended. This restriction is required because element names are used to identify the particular extension mapping used when unmarshalling, and abstract mappings have no fixed element name. It means that abstract mappings can never be the "leaf" mappings in an extension structure.

Mappings may use ordered child definitions (where the XML

components bound to the child definitions occur in a particular sequence) or unordered definitions (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 mappings only optional components are allowed as child definitions.

Mapping definitions can be nested, so that the enclosed mapping definitions are only effective within the scope of the enclosing definition. This approach is fine for abstract mappings, but if used for non-abstract mappings it can have a negative impact on performance. JiBX 2.0 will prohibit nesting of non-abstract mappings, so it's best to structure your binding definitions with all non-abstract mappings as direct children of the root element.

The **mapping** element supports three unique attributes along with several common attribute groups, listed below.

Attributes

class — This required attribute gives the fully qualified class name for the object type handled by this mapping.

abstract — Optional flag for an abstract mapping that can be used directly as a type mapping. This must be "true" (abstract mapping) or "false" (normal mapping, the default if the attribute is not specified).

extends — Optional attribute giving the fully qualified class name of a base mapping for which this mapping can be substituted. Generally this is used with mappings for classes which are subclasses of the class handled by the base mapping, but it can be also be used with unrelated classes. The base mapping may itself extend another mapping, in which case the new extensions also become substitutes for that "inherited" base mapping. Extension is based on element names and does not necessarily indicate any kind of inheritance in terms of the Java class structures (though it is often used for the purpose of representing in XML a tree of subclasses which can be used as instances of a base type).

type-

name	Optional attribute giving the type name for an abstract mapping. This allows alternative abstract mappings to be defined for the same class, and referenced by name with a **map-as** attribute on a **structure** element (or looked up at runtime when **force-classes="true"** is used on the **binding** element). As of JiBX 1.1, the value of this attribute is interpreted as namespace qualified.
style	A **value-style** attribute present on the **mapping** 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 object type handled by this mapping. The name is required on non-abstract mappings unless a custom marshaller/unmarshaller is supplied (see the **object** attribute group, below), in which case it is optional. Abstract mappings should not define a name, but are currently allowed to do so with a warning for support of legacy bindings. See the name attribute group description for name 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.
structure	Attributes from the structure group define ordering and reuse of child binding components. See the structure attribute group description for usage details.

mapping elements are subject to the restriction that multiple mappings with the same object type cannot be defined in the same context.