<collection> Element Definition

The **collection** element defines the binding for a Java collection. Many variations of list-like collections are supported, including user-defined collection types. Arrays are also supported. Maps are not supported directly in JiBX 1.0 (but see the JiBX extras description for custom marshaller/unmarshaller classes which can help with this types of structures).

Collections may consist of a single type of object or multiple types. The simplest form of collection just uses a single mapped item type. This may either be defined using the **item-type** attribute, in which case no child definitions are necessary, or implied by the absence of child definitions (equivalent to using **item-type="java.lang.Object"**).

Collections consisting of multiple types are defined using multiple child definitions within the **collection** element. These may be ordered (where the different types occur in a particular sequence) or unordered (where all are mixed together), as determined by the **ordered** attribute of the structure attribute group. Child definitions within collections must define elements rather than text or attribute values (though the elements defined may themselves have attributes and/or text values). Child definitions within collections are always treated as optional, with any number of occurrances of the corresponding element allowed (zero or more).

The **collection** element supports several unique attributes along with several common attribute groups, listed below. The unique attributes are used for special types of data access to the collection. These are all optional, and have defaults for common collection types, including `java.util.Vector` and `java.util.ArrayList` (as well as subclasses of these classes) along with collection classes implementing the `java.util.Collection` interface. You can use these attributes to select methods of the containing object for accessing data within a collection (with no property definition for the actual collection object).

One potential issue in working with collections is that JiBX generally needs a way to create an instance of the collection when unmarshalling. If the collection is defined using an interface such as `java.util.List` you'll need to either define a concrete implementation type with a no-argument constructor to be used for

the collection (using the **type** attribute of the property attribute group) or use the **factory** attribute of the object attribute group to define a factory method to call when an instance of the collection is needed. The `org.jibx.runtime.Utility.arrayListFactory` is an example of such a factory method, which can be used directly to supply instances of the `java.util.ArrayList` class.

As with all object-valued properties, if the collection property is already initialized when JiBX begins unmarshalling the existing collection instance will be used to hold the unmarshalled items of the collection. JiBX does not clear items from the existing collection before unmarshalling, so if you want to reuse existing data structures with a collection you should clear the collection yourself before unmarshalling (one easy way of doing this is with a pre-set method on the containing object class). This is only necessary when reusing objects, not when unmarshalling to a new instance of an object (where any collections created by the object constructor will initially be empty in any case). If an element name is used with an optional collection, and that name is missing from an input XML document, the collection property will be set to `null`. If you don't want the collection property to ever be set to `null`, use a wrapper **structure** element for the optional element name around the **collection** element.

Attributes

load-method
: This is an indexed load item method for the collection. If used, the value must be the name of a member method of the collection class taking a single `int` argument and returning the indexed item value from the collection (which must be an instance of `java.lang.Object` unless the **item-type** attribute is used to specify the type of items in the collection). This attribute is only allowed in combination with **size-method**. The generated code will use the specified method for loading values from the collection when marshalling.

size-method
: This is an item count method for the collection. If used, the value must be the name of a no-argument member method of the collection class returning an `int` value giving the number of items in the collection. This attribute is only allowed in combination with **load-method**. The generated

	code will use the specified method for finding the count of items present in the collection when marshalling.
store-method	This is an indexed store item method for the collection. If used, the value must be the name of a member method of the collection class taking an `int` argument and a `java.lang.Object` argument (or the type given by the **item-type** attribute, if present), with no return value. The generated code will use the specified method for storing values to the collection when unmarshalling..
add-method	This is an append item method for the collection. If used, the value must be the name of a member method of the collection class taking a single `java.lang.Object` argument (or the type given by the **item-type** attribute, if present). Any return value from the method is ignored. The generated code will use the specified method to append values to the collection when unmarshalling..
iter-method	This is an iterator method for the collection. If used, the value must be the name of a member method of the collection class taking no arguments and returning a `java.lang.Iterator` or `java.lang.Enumeration` object for the items in the collection. The generated code will use the specified method to iterate through the values in the collection when marshalling.
item-type	If this attribute is used it must be the fully-qualified class name for items contained in the collection. If the specified type is an interface or a class with subclasses any of the implementations of that type can be used in the collection. The default is `java.lang.Object`, allowing any type of objects to be present in the collection.
style	A **value-style** attribute present on the **collection** 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 collection as a whole. The element defined in this way will be a wrapper for the XML representations of all item values from the collection. The name is optional unless a marshaller or unmarshaller is defined (see the **object** attribute group, below), in which case it's forbidden. See the name attribute group description for usage details.
object	Attributes from the object group define the way the collection object is 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.