WoodyBinding

The binding framework

Likely you will want to use Woody to "edit stuff", such as the properties of a bean or data from an XML document (we'll simply use the term object to refer to either of these). This supposes that before you show the form, you copy the data from the object to the form, and after the form has been validated, you copy the data in the form back to the object. To avoid having to write actual code for this, a binding framework is available. The basic definition of a binding is as follows (if you don't know Java, just ignore this):

```
public interface Binding {
    public void loadFormFromModel(Widget frmModel, Object objModel);
    public void saveFormToModel(Widget frmModel, Object objModel);
}
```

A binding can work with any object and can perform the binding in any possible way. Currently one implementation is available, based on JXPath. JXPath allows to address data in both beans and XML documents using XPath expressions, so this binding implementation can be used both with beans and XML documents. The rest of this document will focus on this implementation.

The binding is configured using an XML file. This XML file contains elements in the `wb` namesspace (Woody Binding):

```
xmlns:wb="http://apache.org/cocoon/woody/binding/1.0"
```

What does a binding file look like?

To give you an idea of what a binding file looks like, below a very simple example is shown.

```
<wb:context xmlns:wb="http://apache.org/cocoon/woody/binding/1.0" path="/" >
    <wb:value id="firstname" path="firstName"/>
    <wb:value id="lastname" path="lastName"/>
    <wb:value id="email" path="email"/>
</wb:context>
```

The `id` attribute identifies the WoodyWidgetReference. The `path` attribute is the address of the items in the target object (a Javabean or an XML document). The paths can be arbitrary JXPath expressions.

[Convention] Let's call all elements in the `wb` namespace "binding elements". They all cause a binding-related action to be performed.

The `wb:context` element changes the JXPath context to the specified path. The path expressions on the binding elements occuring inside the context element will then be evaluated in this context, thus relative to the path specified on the wb:context element.

The `wb:value` element is used to bind the value of a widget.

The binding framework can do much more than what is shown in the simple example above, so read on for more meat.

Quick reference of supported binding elements

Element	Description	Attributes	Child elements
wb:*	common settings for all bindings	direction	not applicable, see specific elements
wb:context	changes the JXPath context	path	any
wb:value	binds the value of widgets	id, path	wb:on-update, wd:convertor
wb:aggregate	binds aggregatefield widgets	id, path	wb:value
wb:repeater	binds repeater widgets	id, parent-path, row-path, unique-row-id (deprecated), unique-path (deprecated)	wd:convertor (deprecated), wb:on-bind, wb:on-delete-row, wb:on-insert-row, wb:unique-row
wb:unique-row	specifies unique fields for a repeater row	none	wb:unique-field
wb:unique-field	specifies unique field for a repeater row	id, path	wd:convertor
wb:set-attribute	sets an attribute to a fixed value	name, value	none
wb:delete-node	deletes the current context node	none	none
wb:insert-node	insert a node in an XML document	src, xpath	piece of XML that should be inserted

wb:insert-bean	inserts an object in a list-type bean property	classname, addmethod	none
wb:simple-repeater	binds repeater widgets	id, parent-path,row-path, clear-before-load, delete-parent-if-empty	any
wb:javascript	write binding logic in Javascript	id, path	wb:load-form, wb:save-form
wb:custom	write binding logic in Java	id, path, class, builderclass,factorymethod	wb:config

Detailed reference of binding elements

wb:*/@direction

All Bindings share the ability to have the two distinct actions they provide (i.e. load and save) been enabled or disabled by setting the attribute direction to one of the following values:

value	load active?	save active?
both(default)	yes	yes
load	yes	no
save	no	yes

The default value 'both' for this attribute makes its use optional.

NOTE: this setting replaces the @readonly attribute that was available only on selected bindings.

wb:context

Attributes:

- path
- direction (optional)

Child elements: any

The `wb:context` element changes the JXPath context to the specified path. The path expressions on the binding elements occuring inside the context element will then be evaluated in this context, thus relative to the path specified on the `wb:context` element.

The `wb:context` element is usually used in two occasions. First of all, it is used as the root element of the binding file; because an XML file must always have one root element, and you will usually want to perform more than one binding action.

Secondly, you use `wb:context` if you need to address multiple items having a common base path. On the one hand, this saves you on typing and helps readability, and on the other hand, this improves the performance of the binding. To illustrate this with an example, instead of doing this:

```
...
<wb:value id="firstname" path="info/person/firstName"/>
<wb:value id="lastname" path="info/person/lastName"/>
...
```

it is better to do this:

```
...
<wb:context path="info/person">
  <wb:value id="firstname" path="firstName"/>
  <wb:value id="lastname" path="lastName"/>
</wb:context>
...
```

wb:value

Attributes:

- id
- path
- direction (optional)

Child elements:

- wb:on-update (optional)
- wd:convertor (note the wd: namespace!) (optional)

This binding element is used to bind the value of a widget.

The `wb:on-update` element (which itself has no attributes), can contain one or more binding elements that will be executed if the value of the widget has changed, and thus if the object has been updated. For example, you could use the `wb:set-attribute` binding to set the value of an attribute `changed` to `true`.

The `wd:convertor` element has the same purpose as the `wd:convertor` element in the form definition: it converts between objects (numbers, dates) and strings. This is mostly used when binding to XML documents. Suppose you have defined a certain widget in a form definition to have a "date" datatype, and you want to bind to an XML document which contains the date in the XML Schema date representation, then you could define a convertor as follows:

```
<wb:value id="birthday" path="person/birthday">
  <wd:convertor datatype="date" type="formatting">
    <wd:patterns>
      <wd:pattern>yyyy-MM-dd</wd:pattern>
    </wd:patterns>
  </wd:convertor>
</wb:value>
```

The datatype attribute on the `wd:convertor` element, which you don't have to specify in the form definition, identifies the datatype to which the convertor belongs.

wb:aggregate

Attributes:

- id
- path
- direction (optional)

Child elements:

- wb:value elements

The `wb:aggregate` element is used to bind aggregatefields. Remember that aggregatefields are a special type of widget that groups multiple field widgets and lets the user edit their values in one textbox, splitting the values out to the different widgets on submit based on a regexp.

The `wb:aggregate` binding allows to bind the values of the individual field widgets out of which an aggregatefield widget consists. The bindings for these field widgets are specified by the `wb:value` child elements.

wb:repeater

Attributes:

- id
- parent-path
- row-path
- unique-row-id (deprecated)
- unique-path (deprecated)
- row-path-insert (optional)
- direction (optional)

Child elements:

- wb:identity
- wd:convertor (deprecated)
- wb:on-bind
- wb:on-delete-row
- wb:on-insert-row

NOTE: The attributes `unique-row-id` and `unique-path` and the child element `wd:convertor` are deprecated in favor of `<wb:unique-row>`.

The `wb:repeater` binding binds repeaters based on the concept that each row in the repeater is identified by one or more widgets uniquely. This unique identification is necessary to know which rows in the repeater correspond to which objects in the target collection. Newly added rows in the repeater can (but should not) have a null value for this identification widget(s). Typically this/these widget(s) will not editable, so in most cases it will be an output widget. If you don't need the identification widget(s) at the client you don't need to add them to the template at all! You only have to specify `direction="load"` to this/these widget(s) then. This prevents the database IDs from getting to the client.

The `id` attribute should contain the id of the repeater.

The `unique-row-id` attribute specifies the id of the widget appearing on each repeater row that contains the unique identification for that row. The `unique-path` attribute contains the corresponding path in the object model.

NOTE: Both attributes are deprecated. Please use <`wb:identity`> instead.

The `parent-path` and `row-path` attributes can best be understood when described differently for XML documents and Javabeans.

For XML documents:
If you have an XML structure like this:

```
<things>
   <thing ... />
   <thing ... />
</things>
```

then the parent-path attribute contains the path to the containing element ("things") and the row-path attribute contains the path to the repeating element ("thing").

For beans:
if your bean has a property "things" which is a Collection [or whathever JXPath supports as lists], then the parent-path should simply contain "." and the row-path "things".

For both beans and XML documents there is an optional attribute row-path-insert which functions just like the row-path but is used for the nested on-insert-row binding (see below). By default the row-path-insert just takes the value of the row-path. By explicitely setting them different one can exploit one of the following use cases:

- (1) use xpath-predicates in the row-path (note that you can not do that on the row-path-insert)
- (2) save the inserted rows in a different target-node of the backend model.

A child element `wd:convertor` can be used to specify the convertor to use in case the unique-id from the model is a String (typical for XML documents) and the matching widget inside the repeater has a different type.

NOTE: This element is deprecated at that place as it is only used in combination with the deprecated attributes `unique-row-id` and `unique-path`. Please use <`wb:identity`> instead.

The three remaining child elements `wb:on-bind`, `wb:on-delete-row`, `wb:on-insert-row` should contain the binding elements that have to be executed in case of these three events.

The children of the `wb:on-bind` element are executed when an existing repeater row is updated, or after inserting a new row. The JXPath context is automatically changed to match the current row.

The children of the `wb:on-delete-row` element are executed when a repeater row has been deleted. If you want to delete the row, then put a <`wb:delete-node/`> in there. Alternatively, you could also use the `wb:set-attribute` binding to set e.g. an attribute `status` to `deleted`.

The children of the `wb:on-insert-row` are executed in case a new row has been added to the repeater. Typically this will contain a `wb:insert-node` or a `wb:insert-bean binding` (see the descriptions of these binding elements for more details).

The childrens of the `wb:unique-row` specify the widgets appearing on each repeater row for the unique identification of that row. Each <`wb:unique-field`> child specifies one widget.

wb:identity

Child elements:

- wb:value widget-bindings that make up the identity

The <`wb:identity`> is just a container for the child elements specifying the bindings of the identification widgets.

The nested elements just describe regular value bindings that can declare their own convertor if needed.

NOTE: This 'identity' binding is only active in the 'load' operation, so specifying the `direction="save"` is meaningless.

wb:set-attribute

Attributes:

- name
- value
- direction (optional)

Child elements: none

Set the value of the attribute specified in the name attribute to the fixed string value specified in the value attribute.

NOTE: This binding is never active in the 'load' operation, so there is no need to specify the `direction="save"` to protect you model from being changed during load.

wb:delete-node

Attributes:

- direction (optional)

Child elements: none

Deletes the current context node.

NOTE: This binding is never active in the 'load' operation, so there is no need to specify the `direction="save"` to protect you model from being changed during load.

wb:insert-node

Attributes:

- src (optional)
- xpath (optional, only in combination with src)
- direction (optional)

Child elements: the piece of XML that should be inserted

This binding element can only be used when the target object is an XML document (DOM-tree).

It inserts the content of the `wb:insert-node` element as child of the current context element, or, if a src attribute is specified, retrieves the XML from the specified source and inserts that as child of the current context element. In this last case, you can also supply an xpath attribute to select a specific element from the retrieved source.

NOTE: This binding is never active in the 'load' operation, so there is no need to specify the `direction="save"` to protect you model from being changed during load.

wb:insert-bean

Attributes:

- classname
- addmethod
- direction (optional)

This binding element can only be used when the target object is a Javabean.

It instantiates a new object of the type specified in the classname attribute and calls the method specified in the addmethod attribute on the current context object with the newly instantiated object as argument.

NOTE: This binding is never active in the 'load' operation, so there is no need to specify the `direction="save"` to protect you model from being changed during load.

wb:simple-repeater

Attributes:

- id
- parent-path (same as in wb:repeater)
- row-path (same as in wb:repeater)
- clear-before-load (default true)
- delete-parent-if-empty (default false)
- direction (optional)

Child elements: any

A simple repeater binding that will replace (i.e. delete then re-add all) its content.

Works with XML or with JavaBeans if a JXPath factory is set on the binding context.

wb:javascript

Attributes:

- id
- path
- direction (optional)

Child elements:

- wb:load-form
- wb:save-form

Specifies the binding using two JavaScript snippets, respectively for loading and saving the form.

Example:

```
<wb:javascript id="foo" path="@foo">
  <wb:load-form>
    var appValue = jxpathPointer.getValue();
    var formValue = doLoadConversion(appValue);
    widget.setValue(formValue);
  </wb:load-form>
  <wb:save-form>
    var formValue = widget.getValue();
    var appValue = doSaveConversion(formValue);
    jxpathPointer.setValue(appValue);
  </wb:save-form>
</wb:javascript>
```

This example is rather trivial and could be replaced by a simple `<wb:value>`, but it shows the available variables in the script:

- `widget`: the widget identified by the `id` attribute,
- `jxpathPointer`: the JXPath pointer corresponding to the `path` attribute,
- `jxpathContext` (not shown): the JXPath context corresponding to the `path` attribute

It's much more interesting to fill a selection list via `wb:javascript` as there is no built-in element for it at the moment. Imagine your binding bean contains a collection field:

```
<wb:javascript id="selectionListWidget" path="objectCollection" direction="load">
  <wb:load-form>
    var collection = jxpathPointer.getNode();
    widget.setSelectionList(collection, "id", "name")
  </wb:load-form>
</wb:javascript>
```

NOTE:

- The `<wb:save-form>` snippet should be ommitted if the `direction` attribute is set to `load`.
- The `<wb:load-form>` snippet should be ommitted if the `direction` attribute is set to `save`.
- The `@readonly` attribute supported in early versions of this binding has been replaced by the `@direction` attribute as supported now on all binding elements.

wb:custom

NOTE: this binding was never available in Woody, it was added on the cforms branch when the woody branch got deprecated. Its documentation just sits here waiting for the rest of the woody-docs to switch to cforms.

Attributes:

- id (optional, if not provided the containing widget-context will be passed)
- path (optional, if not provided "." is assumed)
- direction (optional)
- class (optional, if not present @builderclass and @factorymethod should be)
- builderclass (optional)
- factorymethod (optional)

Child elements:

- wb:config

Allows to specify your own user-defined binding to be written in Java. There are two essential modes of operation reflected in two examples:

Example 1 - No configuration required:

```
<fb:custom id="custom" path="custom-value"
    class="org.apache.cocoon.forms.samples.bindings.CustomValueWrapBinding"/>
```

This describes the classname of your user defined binding class.

Above imposes the following requirements:

1. there is a `class` CustomValueWrapBinding available in the specified package
2. it has a default (i.e. no arguments) constructor
3. it is a subclass of org.apache.cocoon.forms.binding.AbstractCustomBinding

This last will impose the implementation of two methods:

- void doLoad(Widget widget, JXPathContext context) throws BindingException;
- void doSave(Widget widget, JXPathContext context) throws BindingException;

where the available arguments are

- `widget`: the widget identified by the `id` attribute,
- `context`: the JXPath context corresponding to the `path` attribute

Example 2 - with nested configuration:

```
<fb:custom id="config" path="config-value"
    builderclass="org.apache.cocoon.forms.samples.bindings.!CustomValueWrapBinding"
    factorymethod="createBinding" >
    <fb:config prefixchar="[" suffixchar="]" />
</fb:custom>
```

The additional requirements to your user defined classes are now:

1. there is a `builderclass` CustomValueWrapBinding class having a static `factorymethod`
2. that can (optionally) take an org.w3c.dom.Element holding it's configuration
3. and return an instance of your own user-defined binding which must be a non abstract subclass of org.apache.cocoon.forms.binding.AbstractCustomBinding

Further pointers

- the examples included with woody
- the cocoon-users mailing list