Wicket's XHTML tags

The HTML documents that Wicket uses as templates can contain several special attributes and tags

Using Wicket Tags

To make most HTML editors not complain about using wicket tags and attributes, you need to declare the namespace. You can use any value for the namespace 'xmlns:wicket' but this value should be mapped in your HTML editor to one of the following definitions:

```
Wicket 1.5+ (XML Schema): http://git-wip-us.apache.org/repos/asf/wicket/repo?p=wicket.git;a=blob_plain;f=wicket-core/src/main/resources/META-INF/wicket-1.5.xsd;hb=master
Wicket 1.4 (DTD): http://wicket.apache.org/dtds.data/wicket-xhtml1.4-strict.dtd
Wicket 1.3 (DTD): http://wicket.apache.org/dtds.data/wicket-xhtml1.3-strict.dtd
Wicket 1.2 and earlier: http://wicket.sourceforge.net/
```

Note that some HTML editors (e.g. IntelliJ IDEA's) can't handle the redirect (302) when fetching the schema. Start URL with **https** as workaround.

For example the first lines of a markup file could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:wicket="http://wicket.apache.org/dtds.data/wicket-xhtml1.4-strict.dtd" xml:lang="da" lang="da">
```

or

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:wicket="http://wicket.apache.org/dtds.data/wicket-xhtml1.3-strict.dtd"
      xml:lang="en"
      lang="en">
```

or

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:wicket="http://my.site.org">
```

In the latter demo we use custom value for 'wicket' namespace. To make it working you need to provide a mapping in your HTML editor that will map from 'http://my.site.org' to any of the available XML Schema or DTDs above.
For example in Intellij IDEA you can do that with: File -> Settings -> Schemas and DTDs -> External Schemas and DTDs -> add new entry with URI 'http://my.site.org' and Location one of the above.

This will setup the "wicket" namespace for Wicket tags. You can change the namespace used, for example xmlns:wcn="DTD here" will change it to "wcn" for this markup file only. Without any xmlns declaration the default is "wicket". Just using xmlns:wicket and xmlns:wcn on their own will work as well, as long as you don't have any additional namespace declarations.
What you can not do (but is possible with XHTML) is change the namespace within the very same markup file per tag. Wicket does allow it only with the html tag.

If your IDE needs a DTD, for Wicket 1.3 and later the namespace is also the URL where the DTD can be retrieved.

Wicket Attributes

Attribute wicket:id

wicket:id="ID_HERE" - Used on any element to which you want to add a component.

```
<span wicket:id="hellotext">Hello</span>
```

The value of the attribute is duplicated in the java code:

```
add(new Label("hellotext", "Hello World"));
```

Attribute wicket:message

wicket:message="attribute:resource_identifier" - Used on any tag that we want Wicket to provide an attribute with a value that's the result of a resource lookup.

```
e.g. <input type="submit" wicket:message="value:page.search"/>
=>   <input type="submit" value="Search Page""/>
```

Use comma separator if you have more than one attribute for internationalization:

```
e.g. <img src="some.gif" wicket:message="alt:page.image.alt,title:page.image.title"/>
=>   <img src="some.gif" alt="The Alt" title="The title"/>
```

Attribute wicket:enclosure

```
<tr wicket:enclosure="">
```

The <tr> tag used in the example can be replaced with any html tag that can contain child elements. If the value of the attribute is empty it determines the wicket:id of the child component automatically by analyzing the wicket component (in this case only one wicket component is allowed) in between the open and close tags. If the enclosure tag has a non-empty value for wicket:enclosure attribute like

```
<tr wicket:enclosure="controllingChildId">
```

than more than just one wicket component inside the enclosure tags are allowed and the child component which determines the visibility of the enclosure is identified by the attribute value which must be equal to the relative child id path.

In contrast to <wicket:enclosure> (the tag) the attribute based enclosure can be used for Ajax re-paints.

Attribute wicket:for

<label wicket:for="name" /> - from *AutoLabelResolver* Javadoc:
Resolver that implements the wicket:for attribute functionality. The attribute makes it easy to set up <label> tags for form components by providing the following features without having to add any additional components in code:

- Outputs the for attribute with the value equivalent to the markup id of the referenced form component
- Appends required css class to the <label> tag if the referenced form component is required
- Appends error css class to the <label> tag if the referenced form component has failed validation
- Appends disabled css class to the <label> tag if the referenced form component has is not enabled in hierarchy

The value of the wicket:for atribute can either contain an id of the form component or a path to it using the standard : path separator. Note that .. can be used as part of the path to construct a reference to the parent container, eg ..:..:foo:bar. First the value of the attribute will be treated as a path and the <label> tag's closest parent container will be queried for the form component. If the form component cannot be resolved the value of the wicket:for attribute will be treated as an id and all containers will be searched from the closest parent to the page.

Given markup like this:

```
<label wicket:for="name"><wicket:label>Name</wicket:label>:</label><input wicket:id="name" type="text"/>
```

If the name component has its label set to 'First Name' the resulting output will be:

```
<label for="name5">First Name:</label><input name="name" type="text" id="name5"/>
```

However, if the name component does not have a label set, it will be set to Name based on the markup.

Attribute wicket:unknown

Document me!
Used in org.apache.wicket.markup.html.form.AutoLabelTextResolver.TextLabel#findLabelContent

Attribute wicket:scope

Document me!
Used in org.apache.wicket.markup.html.HeaderPartContainer#getScopeFromMarkup

Wicket Tags

Element wicket:link

<wicket:link> - Support for wicket autolink functionality. Normally, you need to add a model (for example, a BookmarkablePageLink) for each link that Wicket manages. Using the wicket:link tag will automatically do this in the background for you. wicket:link handles static references not only defined by <a> tag, but also by <link>, <script>, <img>, etc.

Example:

```
<wicket:link><a href="Index.html" >Link to wicket document Index.html</a></wicket:link>
```

Now if you request Index.html it gets transformed to:

```
<span><em>Link to wicket document Index.html</em></span>
```

If you want to change the markup to something else you can configure your wicket webapp in the init() method like

```
getMarkupSettings().setDefaultBeforeDisabledLink("<bold>");
getMarkupSettings().setDefaultAfterDisabledLink("</bold>");
```

Which will result into to following:

```
<span><bold>Link to wicket document Index.html</bold></span>
```

Element wicket:panel

<wicket:panel> - The wicket:panel tag surrounds a component. This lets you wrap the component with HTML and BODY tags (so it can be displyed in a browser) but, when you include it, only the content inside the wicket:panel tag is actually included.

Example:

```
<html xmlns:wicket="http://wicket.apache.org">
  <body>
    <wicket:panel>
      PANEL CONTENT HERE
    </wicket:panel>
  </body>
</html>
```

Element wicket:fragment

<wicket:fragment> - is similar to <wicket:panel> but its is declared in the parent's markup instead of in a separate markup file.
Example:

"MyPage.html"

```
<html xmlns:wicket="http://wicket.apache.org">
  <body>
     <span wicket:id="panel1">panel</span>
     <span wicket:id="panel2">panel</span>

     <wicket:fragment wicket:id="frag1">This is the content of fragment 1</wicket:fragment>
     <wicket:fragment wicket:id="frag2">fragment 2222222</wicket:fragment>
  </body>
</html>
```

"MyPage.java"

```
public MyPage() {
  Fragment panel1 = new Fragment("panel1", "frag1", MyPage.this);
  add(panel1);

  Fragment panel2 = new Fragment("panel2", "frag2", MyPage.this);
  add(panel2);
}
```

Elements wicket:border and wicket:body

<wicket:border> and <wicket:body> - (I think this is the same as wicket:panel, except a border is drawn by default; verify).

Element wicket:extend

<wicket:extend> - Extend the markup of the superclass with this content.

Element wicket:child

<wicket:child> - Wicket will replace this content with the markup of the derived component (see <wicket:extend>)

Element wicket:message

<wicket:message> - Wicket will replace this with a string that is retrieved from a resource bundle. e.g.

```
<wicket:message key="page.label">Default label</wicket:message>
```

Starting from Wicket 1.4 you can nest components within a wicket:message element. For example:

```
<wicket:message key="myKey">
  This text will be replaced with text from the properties file.
  <span wicket:id="amount">[amount]</span>.
  <a wicket:id="link">
    <wicket:message key="linkText"/>
  </a>
</wicket:message>
```

```
myKey=Your balance is ${amount}. Click ${link} to view the details.
linkText=here
```

and

```
add(new Label("amount",new Model("$5.00")));
add(new BookmarkablePageLink("link",DetailsPage.class));
```

Results in:

```
Your balance is $5.00. Click <a href="...">here</a> to view the details.
```

Element wicket:remove

<wicket:remove> - Wicket will remove this content in the final markup. This is useful for when you want your web designer to be able to show repeated content when they're working on it, but you want to generate that content using a ListView (or other loop).

Element wicket:head

<wicket:head> - Used for header contributions. Using this, panels, borders and child pages can add header sections to the pages they are placed on. For instance:

```
<wicket:head>
      <script type="text/javascript">
      function foo() {
         alert("Hello, World!");
      }
    </script>
</wicket:head>
<wicket:panel>
   <a href="#" onclick="foo();">Click me!</a>
</wicket:panel>
```

Element wicket:header-items

<wicket:header-items/> - (since 6.15.0) A special placeholder tag that may be used to put all header contributions, either contributed from Java code or from <wicket:head>, in a predefined position in the page's <head> element.

<wicket:header-items> demo

```
 <head>
   <meta charset="utf-8">
   <meta name="someKey" content="someValue">
   <wicket:header-items/>
   <title>My page title</title>
   <link .../>
 </head>
```

With markup like this all header contributions done by using IHeaderResponse or via usage of <wicket:head> will be inserted between <meta name=" someKey"> and <title> elements. This way the application developer can make sure that some head elements, like the special <meta charset="utf-8">, are always rendered before/after Wicket contributions.

Element wicket:enclosure

<wicket:enclosure> - (since 1.3) Useful for markup whose visibility depends on visibility of surrounded component. Lets take a simple example where you want to show a table row, but only if a Label is visible.

```
<tr><td class="label">Phone:</td><td><span wicket:id="phone">phone number</span></td></tr>
<wicket:enclosure>
<tr><td class="label">Fax:</td><td><span wicket:id="fax">fax number</span></td></tr>
</wicket:enclosure>
```

```
add(new Label("fax") { public boolean isVisible() { return getModelObjectAsString()!=null; }});
```

If the label is not visible then neither is the contents of the enclosure. Without the enclosure you would have to add an extra WebMarkupContainer to your code and attach it to the tr tag then link its visibility to that of the label, with wicket:enclosure it is much simpler.

If there is more than one wicket component directly underneath the enclosure, you have to specify which one controls the visibility by providing its id in the enclosure's child attribute:

```
<wicket:enclosure child="address.street">
<tr>
  <td class="label">Address:</td>
  <td><span wicket:id="address.street"></span><span wicket:id="address.city"></span></td>
</tr>
</wicket:enclosure>
```

For nested children, specify the full path, separating them with ":", i.e.

```
<wicket:enclosure child="users:fax">
<table wicket:id="users">
  <tr><td class="label">Fax:</td><td><span wicket:id="fax">fax number</span></td></tr>
</table>
</wicket:enclosure>
```

Note: Changing the visibility of a child component in Ajax callback method will not affect the entire enclosure but just the child component itself. This is because only the child component is added
to the AjaxRequestTarget.

Element wicket:container

<wicket:container> - Sometimes adding components in certain ways may lead to output of invalid markup. For example, lets pretend we output table rows two at a time using a repeater. The markup would look something like this:

```
<table>
     <span wicket:id="repeater">
          <tr><td>...</td></tr>
          <tr><td>...</td></tr>
     </span>
</table>
```

Notice that we had to attach the repeater to a component tag - in this case a span, but a span is not a legal tag to nest under table. So we can rewrite the example as following:

```
<table>
     <wicket:container wicket:id="repeater">
          <tr><td>...</td></tr>
          <tr><td>...</td></tr>
     </wicket:container>
</table>
```

The above is valid markup because wicket namespaced tags are allowed anywhere.

Note that you cannot use HTML markup IDs on <wicket:container> elements. E. g. you cannot replace such an element via AJAX. Also, <wicket: container> elements are completely removed from produced markup when Wicket is run in production mode.