XBL

Laurent Jouanneau
Course on Mozilla Education and Technologies @ Evry
December, 2011

History

- Born in Mozilla years ago
- No official specification. The Mozilla's implementation can be called « version 1.0 ». No other implementation.
- A new specification, XBL 2.0, has been written by the WHATWG and was a candidate recommandation in the W3C
- However, implementors didn't like it. XBL2 is dead (since sept 2011). It has been decided to split the specification into several specifications : shadow dom, Api to define components, etc...

What is XBL ?

- This is an XML language to describe a UI component :
  - Its appearance and content
  - Its behaviors
- It's used to implement many of XUL elements
- You can use it to create your own elements or to enhance an existing one
- You can use XBL to create your own XML languages for interfaces (like XForms with XBL2)

Advantages

- It allows to keep the main document lighter
- Facilitates the development: content and behaviors are separated
- Reusable
- Lighter and fastest solution than pure javascript components, and more features

How it works

- XBLs are described in some files or inline
- XBL are attached to elements through the css property -moz-binding
- Behaviors are developped with javascript
- Can be localized (DTD, etc)
- inheriting

XBL skeleton

```
<?xml version="1.0"?>
<bindings id="numericboxBindings"
   xmlns="http://www.mozilla.org/xbl"
   xmlns:html="http://www.w3.org/1999/xhtml"
   xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
   xmlns:xbl="http://www.mozilla.org/xbl">

  <binding id="numericbox">
    <!-- definition of a first binding -->
    <content> </content>
    <implementation> </implementation>
    <handlers> </handlers>
    <resources> </resources>
  </binding>

  <binding id="other">
    <!-- definition of a second binding -->
  </binding>

</bindings>
```

XBL skeleton II

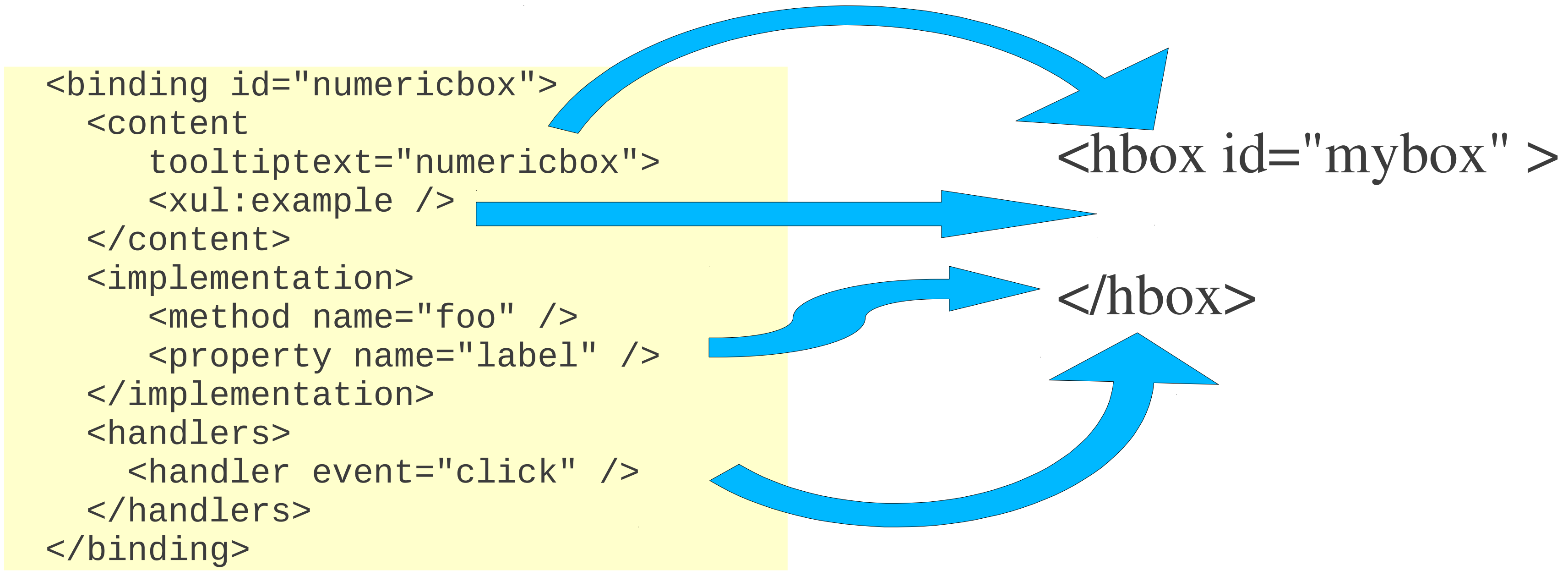
- <content> : defines the XUL content of the binding
- <implementation>: defines methods and properties
- <handlers>: declares event handlers
- <resources>: specify CSS style sheets to attach to the binding

Attachment

- -moz-binding CSS property
- Only one binding at a time on an element
- Life duration: because it is attached with a css property, when the element is removed from the document, the binding is detached

Attachment (2)

hbox#mybox { -moz-binding:url(example.xml#numericbox);}



```
<binding id="numericbox">
  <content
     tooltiptext="numericbox">
     <xul:example />
  </content>
  <implementation>
     <method name="foo" />
     <property name="label" />
  </implementation>
  <handlers>
    <handler event="click" />
  </handlers>
</binding>
```

<hbox id="mybox" >

</hbox>

Attachment (3)

Equivalent of the result (we will saw later real differences with this dom tree: the <example> element is in fact an anonymous node):

```
<hbox id="mybox" tooltiptext="numericbox">
  <example />
</hbox>
```

```
var box = document.getElementById('mybox');
box.foo();
box.label;
```

Warning: because the attachment is made with a css property, it is made asynchronously. Methods, properties etc are not available immediately.

Content

- Any elements in <content /> (xul, xhtml...)
- Defined content is anonymous: elements are not attached in the document like all original elements of the document
- xbl:inherits: inheritance of attributes
- xbl:text in xbl:inherits: text node generation
- Inserting of non anonymous content: <children />

Simple content

Example of a simple button (simple_button.* files)

simple_button.xml

```
  <binding id="simple-button">
    <content>
      <xul:hbox align="center" pack="center" flex="1">
          <xul:image src="process-stop.png"/>
          <xul:label value="Stop"/>
      </xul:hbox>
    </content>
  </binding>
```

simple_button.xul

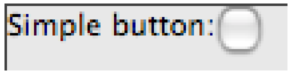
```
<hbox>
   <description>Simple button:</description>
    <simplebutton id="mybutton"/>
</hbox>
```

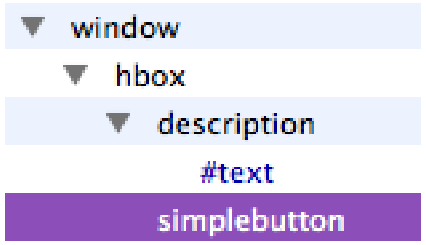
simple_button.css

```
simplebutton {
  -moz-appearance:button;
  -moz-binding:url(simple_button.xml#simple-button);
}
```

Simple content (2)

Without -moz-binding

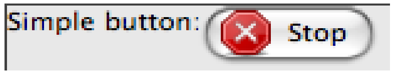
xul: 

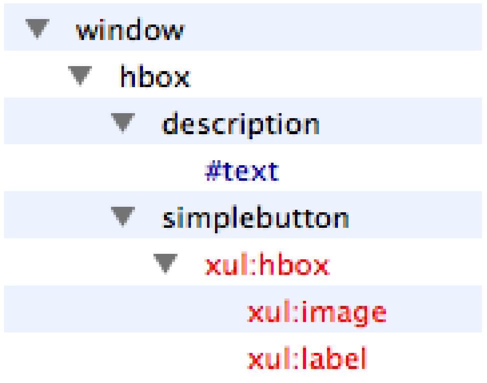
The DOM tree: 

```
▼ window
  ▼ hbox
    ▼ description
        #text
      simplebutton
```

`document.getElementById('mybutton').childNodes.length` Returns 0

With -moz-binding

xul: 

The DOM tree: 

```
▼ window
  ▼ hbox
    ▼ description
        #text
    ▼ simplebutton
      ▼ xul:hbox
          xul:image
          xul:label
```

`document.getElementById('mybutton').childNodes.length`

Returns 0 too ! Because added nodes are **anonymous**

Content + child nodes

simple_button2.xml

```
  <binding id="simple-button">
    <content>
      <xul:hbox align="center" pack="center" flex="1">
          <xul:image src="process-stop.png"/>
          <xul:label value="Stop"/>
          <children />
      </xul:hbox>
    </content>
  </binding>
```

simple_button2.xul

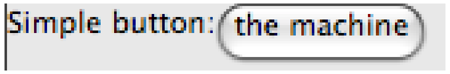
```
<hbox>
   <description>Simple button:</description>
    <simplebutton id="mybutton">
       <description>the machine</description>
    </simplebutton>
</hbox>
```

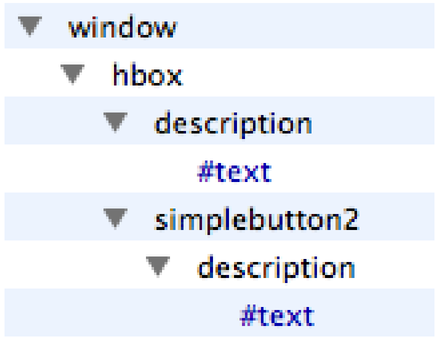
simple_button2.css

```
simplebutton {
  -moz-appearance:button;
  -moz-binding:url(simple_button2.xml#simple-button);
}
```

Content + child nodes (2)

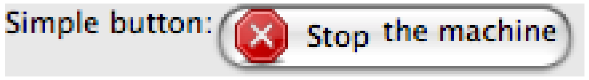
Without -moz-binding

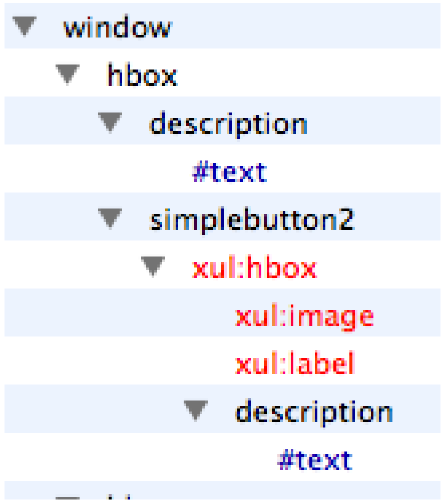
xul: 

The DOM tree: 

`document.getElementById('mybutton').childNodes.length` Returns 1

With -moz-binding

xul: 

The DOM tree: 

`document.getElementById('mybutton').firstChild.localName`

Returns « description »

Anonymous content

- Elements inside <content> are called « anonymous content » and are part of the « shadow tree »
- Explicit content (children of the bound element) replace the <children> element, but remain as direct children of the bound element
- From the point of view of the DOM document, anonymous content is « invisible »
- To access to the anonymous content :
  - document.getAnonymousElementByAttribute
  - document.getAnonymousNodes

More inserting point

```
<binding id="simple-button">
  <content>
    <xul:hbox align="center" pack="center" flex="1">
        <xul:image src="process-stop.png"/>
        <children includes="label|description">
           <xul:label value="Stop"/>
        </children>
    </xul:hbox>
    <xul:vbox>
        <children />
    </xul:vbox>
  </content>
</binding>
```

The « includes » attribute indicate which type of element could be inserted.
The content of <children> is used when there are no children. This is the default content.

Content: attributes inheritance

With the xbl:inherits attribute, an anonymous element can have attributes inherited from the bound element.

```
<binding id="simple-button">
  <content>
    <xul:hbox  xbl:inherits="align,dir,pack,orient"
         align="center" pack="center" flex="1">
        <xul:image src="process-stop.png" xbl:inherits="src=image"/>
        <xul:description xbl:inherits="xbl:text=label,accesskey,crop" />
        <children />
    </xul:hbox>
  </content>
</binding>
```

```
<simplebutton3 id="mybutton" label="Launch" image=""
                    pack="start" orient="vertical">
   <description>the machine</description>
</simplebutton3>
```

The value of the label attribute will be stored as a text node of the description element. The value of the image attribute will be stored in the src attribute of the image element. The hbox element will inherit the pack and the orient attribute.

Implementation

- In the <implementation> element, methods and properties can be defined
- This methods and properties are then available on the bound element
- You can define precisely:
  - Constructor / destructor
  - Methods
  - Properties
  - Fields
- The « implements » attribute can be set to indicate the list of interfaces that the binding implements.

Implementation: constructor and destructor

- Constructor:
  - Called during the attachement, when the property -moz-binding is applied
  - No parameters. Parameters should be passed through attributes on the bound element
- Destructor: called during the detachment, when the CSS property is not applied anymore

```
<constructor><![CDATA[
   // the code
]]></constructor>
<destructor><![CDATA[
   // the code
]]></destructor>
```

Implementation: properties

- <field> element
- <property> element on which you can define getters and setters
  - Onget attribute or <getter> element
  - Onset attribute, or <setter> element
- Read only : attribute readonly="true"

```
<field name="mValue">'javascript value'</field>

<property name="example" onget="return this.mValue;">
   <setter> this.mValue = val; </setter>
</property>
```

Implementation: methods

- <method> element
- Parameters with the <parameter> element
- <body> element to define the content of the method

```
<method name="example">
  <parameter name="aParameter" />
  <body><![CDATA[

    // the code
     var foo = aParameter * 3;
    this.otherMethod();

  ]]></body>
</method>
```

Handlers

- To attach event handler on the bound element, use <handler> elements in the <handlers> section
- Attributes:
  - Event: the event name
  - Phase: the phase of the event (capturing, bubbling, target)
  - Button, clickcount: for mouse events
  - Keycode, charcode, modifiers: for key events
- The javascript code of the handler should be store in the action attribute, or as the content of the handler element

handlers

Example:

```
<handler event="keypress" keycode="VK_LEFT" modifiers="alt"
         action="/*here the code*/" />

<handler event="click" phase="capturing" button="2">
<![CDATA[

// here the code

]]></handler>
```

Ressources

- To add stylesheets applied on the anonymous content
- Only the shadow tree of the bound element is affected by the CSS properties of the stylesheet. CSS styles are not applied on the explicit content.
- Note that the shadow tree is affected by CSS styles of the document, unless inheritstyle="false" is set on <binding>

```
<resources>
  <stylesheet src="styles.css" />
</resources>
```

Inheritance

- An binding can inherit from an other binding
- Indicate the url of the parent binding into the « extends » attribute on the <binding> element.
- <content>, methods and properties defined in the child binding overload existing content, methods or properties of the parent binding. In this case, methods of the parent binding cannot be called, except the constructor and the destructor
- constructors of parents are called before the constructor of the binding child.

Example: a slideshow

- See slideshow.xml
- This binding allow to create easily a slide system : put your « slides » into this element, and it will display each one at a time. It provides also some buttons to navigate.

Debuging

- The XBL content is not shown?
  - Vérify that the stylsheet is applied
    - <?xml-stylesheet ?> ?
    - path ok ? Css selector ok ?
    - DOM inspector → verify styles of the element, its xbl properties etc..
  - Error console : check if there are errors
    - XML errors (malformed XML, encoding...)
    - JS parsing errors in methods
  - Content : XUL elements with XUL namespace?