

FIREFOX HACKS™

*Tips & Tools for Next-Generation
Web Browsing*



O'REILLY®

Nigel McFarlane

HACK
#69

Make New Tags and Widgets with XBL

Don't like the set of HTML or XUL tags that browsers provide? Make new ones.

The HTML and XHTML standards allow a content author to extend the set of composition elements on a web page by applying special styles to the `<div>` and `` tags and their content. A styled `<div>` block might appear to be a navigation bar, an advertisement, or some other specialized content that HTML does not explicitly define. XML Binding Language (XBL) is an alternate strategy for naming features of web pages. This hack shows how you might use it.

XBL is a Mozilla technology that's slowly being adopted by the W3C. It first appeared at the W3C as a technical note at <http://www.w3.org/TR/2001/NOTE-xbl-20010223/>. More recently, it has entered the formal standardization track via the sXBL standard (the *s* indicates that XBL's initial standard use is in SVG). That standard is here: <http://www.w3.org/TR/sXBL>. Mozilla XBL syntax is not yet the same as the W3C syntax, but the concepts are all the same. It will match the standard one day.

In Firefox, XBL is used to create a new tag for any XML or HTML dialect, but most frequently for XUL. Public or strictly conforming HTML and XHTML documents should not be hacked with XBL. Use it for custom, private documents only. An XBL binding written in XML is *attached* to a tag using a Mozilla-specific CSS style:

```
tagname { -moz-binding : url("binding-definition.xml"); }
```

It's also possible to say *the binding is bound to the tag*, but the idea of attachment will get you into far less trouble.

Make a `<sidebar>` element for HTML

HTML and XHTML can achieve a lot when combined with CSS, but they don't contain any direct support for *sidebars*. Sidebars are blocks of text, usually floating on the right side of the screen, that contain information that's an aside to the main flow of the text. Let's make a `<sidebar>` tag using XBL. Here's an HTML equivalent, to begin with:

```
<div style="float : right; background-color : lightyellow;">
  <hr />
  <div style="font-weight : bold;">Sidebar Title</div>
  <p>This is the content of the sidebar</p>
  <p>The content could be long</p>
  <div style="text-align : right; font-style : italic;">Anon.</div>
  <hr />
</div>
```

Using XBL, we extract the structural content into a file named *sidebar.xml*:

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
          xmlns:xbl="http://www.mozilla.org/xbl"
          xmlns:xht="http://www.w3.org/1999/xhtml">
  <binding id="sidebar">

    <resources>
      <stylesheet src="sidebar.css"/>
    </resources>

    <content>
      <xht:hr />
      <xht:div class="sidebar-title" xbl:inherits="xbl:text=title" />
      <children />
      <xht:div class="sidebar-author" xbl:inherits="xbl:text=author" />
      <xht:hr />
    </content>

  </binding>
</bindings>
```

The `<content>` part matches the XHTML content, with a few namespaces thrown in. Note the use of XBL's special `inherits` attribute and `<children>` tag mixed into the XHTML. The `<resources>` section is optional. In this case, *sidebar.css* might contain the following styles:

```
.sidebar-title { font-weight : bold; }
.sidebar-author { text-align : right; font-style: italic; }
```

These styles affect the tags *inside* the content part of the binding. These tags can't be styled from HTML. To attach this binding, HTML pages must include extra styles. These styles affect the *whole* binding. The following file, *bind-sidebar.css*, is an example of such a style:

```
sidebar {
  -moz-binding      : url("sidebar.xml#sidebar");
  background-color  : lightyellow;
  float             : right;
}
```

The HTML or XHTML page must include this stylesheet with a `<link>` tag. Once the binding is created, a web page can reuse the structural content freely:

```
<html>
  <head>
    <link rel="stylesheet" href="bind-sidebar.css" type="text/css">
  </head>
  <body>
    <p>
```

```
<sidebar title="First Sidebar" author="Anon.">
  Foo Content
  <br>
  Bar Content
</sidebar>

Lorem ipsum dolor sit amet ...

<sidebar title="Second Sidebar" author="- me.">
  Short Content
</sidebar>

Lorem ipsum dolor sit amet ...
</body>
</html>
```

Figure 6-12 shows this HTML page after the bound `<sidebar>` tags are rendered.



Figure 6-12. HTML page with `<sidebar>` tags derived from XBL

Make a Custom XUL Widget

A more typical use of XBL is to increase the set of widgets that XUL offers. XUL's default widget set is defined in a file called `xul.css` in the chrome. You

can increase the set of widgets by using any stylesheet, or you can [hack that file directly \[Hack #75\]](#). Figure 6-13 shows an example of a simple new widget called `checkboxbutton`.

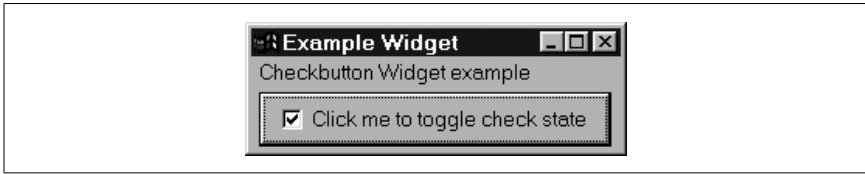


Figure 6-13. XUL window showing XBL `checkboxbutton` widget

Each time you click the button, the small checkbox is toggled on or off. The `checkboxbutton` widget is hooked up to the `<checkboxbutton>` tag using CSS, the same as in the HTML case:

```
checkboxbutton { -moz-binding : url("checkboxbutton.xml#checkboxbutton"); }
```

There are no extra styles in this case. XUL relies on the `<?xml-stylesheet?>` processing instruction to load CSS. The widget has the focus (the dotted line), because it's already been clicked a few times. The tag used to create this example is simply this:

```
<checkboxbutton label="Click me to toggle check state" checked="true"/>
```

Unlike text, widgets are active and responsive, so the `checkboxbutton` XBL binding has more object-like features than the sidebar example. Here's the binding definition:

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl"
  xmlns:xbl="http://www.mozilla.org/xbl" <!-- for xbl:inherits -->
  xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <binding id="checkboxbutton">

    <content>
      <xul:button>
        <xul:checkbox xbl:inherits="checked,label"/>
      </xul:button>
    </content>

    <implementation implements="nsIDOMXULCheckboxElement">
      <property name="checked">
        onget="return this.getAttribute('checked') == 'true';"
        onset="return this.setAttribute('checked', val);" />
      </implementation>

    <handlers>
      <handler event="click"><![CDATA[
        var state = this.getAttribute("checked") == "true" ? 1 : 0;
        this.setAttribute("checked", state ? "false" : "true");
      ]]></handler>
```

```
    </handlers>

    </binding>
</bindings>
```

The `<implementation>` section can define fields, properties, and methods, mostly not illustrated in this example. The `<handlers>` section defines handlers, in this case, a single handler for the `click` DOM 2 event. This handler will fire if no handler is explicitly placed on the `<checkboxbutton>` tag.

Here's a sample XUL document that benefits from this binding:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"?>
<?xml-stylesheet href="bind-check.css"?>

<window
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  title="Example Widget">

  <description>Test widget for the checkboxbutton binding</description>
  <checkboxbutton label="Click me to toggle check state" checked="true"/>
</window>
```