Web menus with XUL and XBL

Finishing Touches

There is certainly a lot packed under the hood of the Mozilla browser. In this article, we will use XUL to create a fine web page menu. Along the way you'll learn about eXtensible Binding Language (XBL) and how you can use XBL to enhance the power of your XUL creations. JONO BACON

In the last few issues of Linux Magazine, we have been exploring the different ways to build interfaces in XUL. In this last part of the series, we will integrate XUL with web content by adding an XUL interface to a web page in the form of a special menu bar. This menu bar will use the same widget set as Mozilla, and it will use the theme and look and feel of the browser. The effect will that of a special toolbar, such as the Google and Yahoo toolbars, but the user will not need to install it specially; it will just load when the user accesses the website.

XUL & the Rest of the Toolbox

With many computer innovations, developers and users put technologies in boxes and discriminate how they should be used. PHP is a good example – although you can certainly use Javascript, ASP and other technologies in PHP pages, the majority of developers seem to stick to PHP and PHP-related technologies.

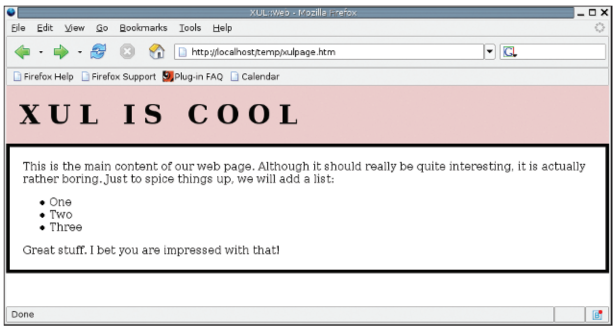
PHP is a server side scripting language; Javascript is a client side scripting language; and HTML is a language for marking up content. Each technology has its defined use, and most developers stick to these uses.

You may have already put XUL into one of these conceptual boxes. In this series we have already looked at examples of XUL scripts such as the Mozilla Amazon Browser, and these scripts seem to crank Mozilla into a full web application mode. These examples demonstrate that the technologies revolve almost entirely around XUL for creating the interface. It is therefore fairly reasonable to put XUL into a conceptual box where it is used for the interface, Javascript is used for interaction, and other technologies don't really get a look in.

Despite the applicability of this concept to some applications, it certainly doesn't hold true for all applications. Javascript has no native support for retrieving information from a MySQL database. How would you get the data from the database into your XUL interface? In a situation like this we would need to resort to a special XPCOM Mozilla object or use another language that has this kind of support.

Another example of integrating XUL with other technologies occurs if you want to integrate XUL elements into your website. What if we wanted to have a special box that displays stock information dynamically? This stock information box is an ideal use for something such as XUL, but we would need to integrate it seamlessly into our website, where HTML and CSS are the orders of the day.

In its current form, XUL can be used virtually anywhere and with virtually any technology. There is some support within Mozilla to help you to perform this kind of integration, but in some cases you need to know a few


XUL::Web - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/temp/xulpage.htm
Firefox Help
Firefox Support
Plug-in FAQ
Calendar
Done

XUL IS COOL

This is the main content of our web page. Although it should really be quite interesting, it is actually rather boring. Just to spice things up, we will add a list:

- One
- Two
- Three

Great stuff. I bet you are impressed with that!

Figure 1: Our simple webpage before we XULify it.

tricks and tips to work around some of XUL's inbuilt limitations.

XUL within a website

With the increasing shift of people moving over to Mozilla from Internet Explorer and other browsers, XUL is becoming viable as a technology that can be used within general web content. Although some web purists may disagree that any specific technology is good for the web, XUL can be used in cases where you are confident that a majority of your users are using Mozilla or where you can offer some content for Mozilla users and content for non-Mozilla users.

Although you could create content specifically for Mozilla users, it is discouraged for the same reason content specifically intended for Internet Explorer users is discouraged. Only create Mozilla-specific content if you *know* Mozilla is used, or if you can create a suitable alternative.

We are going to explore integrating XUL with web content by adding a XUL interface to a web page in the form of a special menu bar for the web site. Not only will this menu bar use the same widget set as Mozilla, but it will use the theme and look and feel of the browser. This can give the same kind of effect of a special toolbar such as the Google and Yahoo toolbars, but users will not need to install it specifically; it will just load when they access the website.

As I mentioned earlier, this article will explore the task of integrating XUL with web content by adding a XUL menu bar to a web page. This kind of use of XUL is called a *Remote Application*. XUL code can theoretically run from a local machine (where you load the page from your hard disk directly with File->Open File) or from a remote web server, where you access the page via a URL such as *http://www.thissite.com/*.

The geographical location of the page does not dictate whether an application is remote – if you are running Apache on your local computer and you access the XUL page via a URL, the application is still remote. The key difference is that a remote application is served to you by a web server, whereas a local application is loaded directly in the browser.

Local and remote XUL applications also differ in terms of functionality. Remote applications have a more limited set of functionality due to potential security risks. An example of this is that a remote application cannot write to a filesystem on the web server. You can get around these limitations by adjusting the configuration of how Mozilla works, but this workaround simply reduces your security and is not suggested in production environments. We are going to focus instead on perfectly acceptable uses of a remote XUL application – clicking on a menu to go to another part of the website.

Creating the code

To add a menu bar to a web page we will need a web page to begin with. You should now all sit back and bask in the glory that is our web page shown in Figure 1. The code below creates this page and you should add it to xulpage.html:

This is the main content of our web page. Although it should really be quite interesting, it is actually rather boring. Just to spice things up, we will add a list:

```
<ul>
          <li>One</li>
          <li>Two</li>
         <li>Three</li>
</ul>
```

Great stuff. I bet you are impressed with that!

To make our page look interesting, we have also created a style sheet in the file *stylesheet.css*. Add this code to the file:

```
body, html {
       margin: 0;
       background: #FFF;
       color: #000;
}

#top
{
       font: verdana, arial;
       font-size: 40px;
       font-weight: bold;
       text-transform: uppercase;
       letter-spacing: 0.3em;
       padding: 20px;
       background: #EACBCB;
}

#content
{
       padding: 20px;
       border: solid thick black;
}
```

With our simple web page ready, we will now add some XUL and discuss how it works. The first step is to actually create the XUL we want to add to our web page. We will add this to a file called *menu.xml*. Although some of the code will be familiar to you, we will be using some special features in Mozilla to bind functionality in chunks of code that can be called from within our HTML. We will go through each line of code to discuss how this all works.

The first two lines state that we are are using XML and specify that the stylesheet being used is with the Chrome registry that is part of Mozilla. This special registry deals with how interfaces can be created and how they look:

```
<?xml version="1.0"?>

<?xml-stylesheet href="chrome:↩
//global/skin/" ↩
type="text/css"?>
```

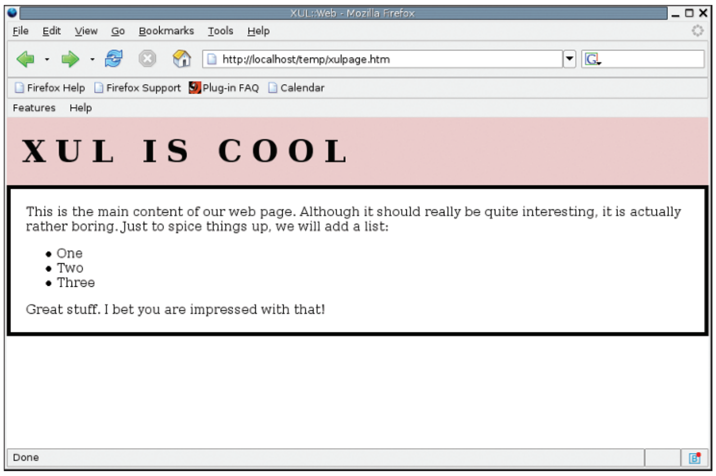
Although XUL is a core technology within the Mozilla suite, there are actually a number of other related

Figure 1 web page

```
<!DOCTYPE HTML PUBLIC "-
//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/lo
ose.dtd">

<html><head>
<title>XUL::Web</title><link
rel="stylesheet"
href="http://localhost/temp/
stylesheet.css" type="text/
css"><link rel="icon"

href="http://mozilla.org/image
s/mozilla-16.png" type="image/
png"></head>

<body>

<div id="top">
XUL Is Cool
</div>
<div id="content">
```

technologies that can be used with XUL to allow applications to be developed. One of these technologies is called the eXtensible Binding Language (XBL).

The purpose of XBL is to simply allow you to create a chunk of functionality and bind it to other content via CSS or the DOM. This gives you the ability to create defined sets of functionality that can be called from your other code – this is not too dissimilar to the concept of functions, but within the XUL/XBL architecture. It should be noted that this is not the same as a function, but its concept is very similar.


XUL::Web - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/temp/xulpage.htm
Firefox Help Firefox Support Plug-in FAQ Calendar
Features Help
Done

XUL IS COOL

This is the main content of our web page. Although it should really be quite interesting, it is actually rather boring. Just to spice things up, we will add a list:

- One
- Two
- Three

Great stuff. I bet you are impressed with that!

Figure 2: Our XUL menu bar added to the page.

To create some of these bindings, we first need to indicate that we are using bindings inside this file. We do this by specifying a *<bindings>* tag and indicating that we are using an XBL namespace. A namespace allows you to define which tags are part of particular technology. If you had a XUL tag called *<menu>* and you had another technology that used a *<menu>* tag, there would be no way of determining which *<menu>* tag you wanted to use.

To resolve this you typically prefix the tag with the technology that you are using (as we will do later for our XUL tags). The *xmlns* attribute in the *<bindings>* tag below indicates that the tags inside this file are within the XBL specification. *<bindings>* itself is not a XUL tag but an XBL tag, as shown in the following example

```
<bindings id="ourmenu"
 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">
```

We will now create a specific binding that refers to a specific XUL layout that we can use. We use the *id* attribute to create a name for our XUL layout. We will use this value to refer to our XUL layout in our HTML code later:

```
  <binding id="ourmenu">
    <content>
```

We now need to begin creating our actual menu bar. To do this, we will create the fairly common menubar, menu, menupopup, menuitem structure we have covered in previous issues.

Here is the code to create a menubar with two menus and a series of menu options. You will see here that we are prefixing each of the tags with *xul* to indicate that these tags are not from the XBL namespace but from the XUL namespace:

The Menu Bar

```
      <xul:menubar oncommand="loadPage(event);">
        <xul:menu label="Features">
          <xul:menupopup>
            <xul:menuitem label="Introducing XUL" value="http://localhost/temp/intro.html" />
            <xul:menuitem label="Why XUL?" value="http://localhost/temp/why.html" />
            <xul:menuitem label="Creating a script" value="http://localhost/temp/createscript.html" />
          </xul:menupopup>
        </xul:menu>
        <xul:menu label="Help">
          <xul:menupopup>
            <xul:menuitem label="General Help" value="http://localhost/temp/generalhelp.html" />
            <xul:menuitem label="Help Index" value="http://localhost/temp/index.html" />
          </xul:menupopup>
        </xul:menu>
      </xul:menubar>
```

You may have noticed that each *<xul:menuitem>* has a *value* attribute that contains a URL. The reason for this is that we want each option in the menu to go to a particular page on the website. Just adding the URL to the value attribute will not do this itself, but we will be using a special *loadPage(event)* function that is referred to on the first line (the *<xul:menubar>* tag). We will define this function later in the file, but all you need to remember is that when an option is selected, the *loadPage (event)* function will be used.

Finally, we will close off the remaining tags:

```
    </content>
  </binding>
</bindings>
```

Although we have not created our *loadPage(event)* function, we can still test our snazzy XUL interface. Before we do this we need to do the all-important job of actually loading the XUL code, or more specifically our XBL binding that contains the XUL code, into our HTML page. Remember that we can use XBL bindings via CSS and the DOM, and we will use CSS for our page.

Right at the top of our *xulpage.html* file, add the following code after the *<body>* tag (and as such before our *top <div>* tag):

```
<div style="-moz-binding: url('menu.xml#ourmenu');"></div>
</div>
```

This code creates a *<div>* area and uses the HTML 'style' attribute to apply a CSS to the tag. This CSS uses the -moz-binding element that is specific to Mozilla, indicating where an XBL binding file is located and which binding you want to use. The common CSS *url()* syntax is used to specify where the file is and what it is called (remember, if you don't specify a path, it is assumed the file is in the same directory as the style sheet – or in our case the HTML, as we are using an inline style here).

We then use *#ourmenu* to indicate that we want to use the *ourmenu* binding from inside that file. This shows how you can have a number of bindings from within the same file.

Now, when you access *xulpage.html* in your browser, you should see something similar to Figure 2.

Creating Functionality

Our XUL is looking great at the top of our web page, but it is not much use if we can't make it do anything interesting. We now need to define the *loadPage* function that we discussed earlier and add some Javascript to make it work. To do this, we can use the *<implementation>* block that XBL provides to flesh out our code. We first need to actually create the block. Create the block after the *</content>* tag and before the *</binding>* tag:

```
<implementation>
```

We now need to define *loadPage*. To do this we use the *<method>* tag. Functions and methods are different terms that describe the same basic concept. We use the *name* attribute to name our method:

```
    <method name="loadPage">
```

When we referenced *loadPage earlier*, notice that there was a single parameter called *'event' (loadPage(event))*. We need to define each parameter in this implementation block separately:

```
    <parameter name="para"/>
```

We are now ready to create some code to define what *loadPage* actually does. We first need to create a *<body>* block to put this code in:

```
    <body>
```

We now write some Javascript that will extract the contents of the *'value'* attribute by using the *getAttribute* DOM method and put the resulting value into a variable called *url*. Note how we refer to parameter name (*para*) as the object we are getting the value from – this is how we get the right URL from whichever menu item was clicked. We then use *document.location* to make the web browser jump to that page:

```
  var url = para.originalTarget↩
  .getAttribute('value');
  document.location = url;
```

Finally, we need to close off all of our remaining tags:

```
  </body>
  </method>
  </implementation>
```

Although it seems our work is complete, the final piece of code we need to add is a handler that will tie together our *oncommand* event where we reference *loadPage(event)* in the XUL code with our XBL binding. You need to add this code immediately after the previous implementation:

```
<handlers>
<handler event="command" action↩
="loadPage(event);"/>
</handlers>
```

Conclusion

This article showed how to use XBL as a middle ground to connect XUL and HTML. This example not only demonstrates how you can use XUL for practical purposes such as providing a site wide menu bar, but it also shows how XBL can be useful for defining chunks of XUL-based functionality and their associated Javascript implementations. With a few tricks, you can bridge together HTML, XBL, XUL, CSS, and Javascript in new and interesting ways. ■

Using PHP with XUL

One of the major problems with XUL at the moment is that there is little support for accessing external services such as MySQL/PostgreSQL databases and bringing this external data into your interfaces. One option is to create an XPCOM object that does this work for you, and some work is going into creating objects that can do this. Although a solution, this is rather complex and it requires you to learn yet another API and programming platform to get your job done. Another option is to use PHP.

One point to bear in mind when using PHP with XUL is that PHP is obviously a server side technology and XUL is client side. With this in mind you don't want to throttle your web server too hard with dynamic requests in XUL. If you do feel you have a reasonable use for PHP in your scripts, you can use it by simply changing the header type of the content. You will need to put your XUL into a .php file so it is processed by the PHP subsystem, and then you can specify in the HTTPD header that the type of content is of the type application/vnd.mozilla.xul+xml (which is XUL). To do this, add the following code at the top of the XUL file:

```
<?php
header( "Content-type: ↩
application/vnd.mozilla.↩
xul+xml" );
?>
```

Within your XUL code you can now use PHP code within the normal PHP <?php and ?> blocks. An example of this could include going through a PHP *while* loop to bring data out of a database and populate a XUL list box. You can then roll in Javascript to dynamically deal with this data on the client side. Adding in this PHP support really brings a wealth of potential to XUL, Javascript and XBL on the client and the server.

INFO

[1] The main Mozilla website: *http://www.mozilla.org/*

[2] XULPlanet: *http://www.xulplanet.com/*

[3] XUL Programmers reference *http://www.mozilla.org/xpfe/xulref/XUL_Reference.html*

[4] XBL Reference: *http://www.mozilla.org/projects/xbl/xbl.html*

[5] XPCOM Information: *http://www.mozilla.org/projects/xpcom/*

[6] Mozilla Amazon Browser: *http://mab.mozdev.org/*

[7] Mozilla XUL based games: *http://games.mozdev.org/*