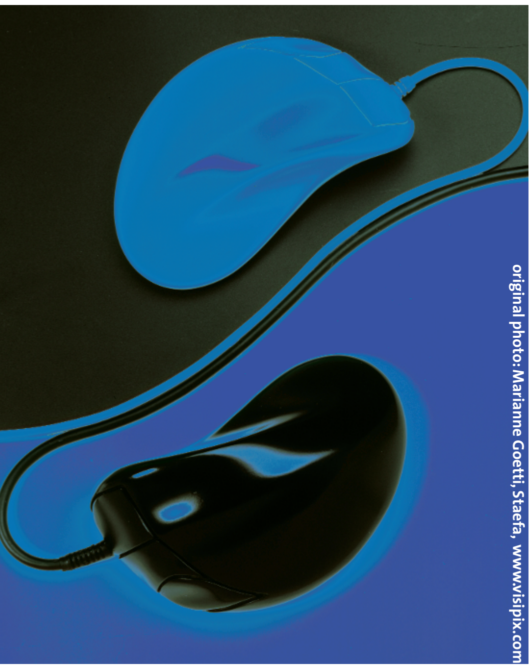
original photo: Marianne Goetti, Staefa, www.visipix.com

Programming with Mozilla XUL

Pretty Clicks

In the first few issues of our XUL series, we have been mainly looking into the simple basics of the language. As our journey has continued, we have explored buttons, labels, geometry management and last month we started to connect functionality to our interface. BY JONO BACON

This month we are going to have a detailed look at what menus are used for and how to create them. We will then tie together the information from the past few issues to create a complete XUL user interface. Although this interface will not include the functionality of a complete application, we will be able to cement our knowledge of XUL by creating a usable and practical interface.

Menus provide a means of hiding away functionality so that it can be accessed easily by the user. The way menus work is quite ingenious. If we needed to display a button for each function in an application; we would end up with a hugely cluttered interface.

Menus have a number of different parts. The first is the menu toolbar. This is the space at the top of an application window that houses the different menu items. Within this space we have separate menu items that refer to different categories of options. Finally, within each of these categories we can click on the menu item and show the full popup menu with a series of options that are related to that menu.

If we click on the File menu in a program, we typically have New, Open, Close, Save, Save As and other options. As long as users are aware of the kind of functions they need, they can select the right menu category and access the relevant option.

You should always have a clear idea of the kind of functionality that needs to go into a menu category and ensure that your menus are intuitive enough to persuade a user to click on the right menu category to access the needed option.

Usability has taken a back seat when it comes to organizing menus, and the result of this has been unrelated menu options living in menu categories. As an example, in Mozilla Firefox there is a Work Offline option in the File menu.

Another example is in Microsoft Windows; if you need to shutdown the computer, you click on the Start button and then select Shutdown. Some of you may disagree with these examples, but you should always be aware of usability when creating your menus.

Creating our first menu

To start coding our first menu, you will need to create the normal boilerplate code that is present in virtually every XUL file. Use the code in Listing 1.

We can now create our menu structure. The first item we need to create is a means for our menu to be located at the top of the screen. This container is called a toolbox and we can add it with the <toolbox> tag:

```
<toolbox>
```

Like other geometry management containers, the <toolbox> tag places the code within it in a specific part of the

Listing 1: first.xul

```
<?xml version="1.0"?>
<?xml-stylesheet
   href="chrome://global/skin/"
   type="text/css"?>

<window
    id="test-window"
    title="Test Program"

   xmlns="http://www.mozilla.org/
   keymaster/gatekeeper/there.is.
   only.xul">
```

Listing 2: second.xul

```
<?xml version="1.0"?>
<?xml-stylesheet
   href="chrome://global/skin/"
   type="text/css"?>

<window
    id="test-window"
    title="Test Program"

   xmlns="http://www.mozilla.org/
   keymaster/gatekeeper/there.is.
   only.xul">

<toolbox>
  <menubar id="menubar">
      <menu id="filemenu"
   label="File">
            <menupopup
   id="filepopup">
               <menuitem
   label="New"/>
               <menuitem
   label="Open"/>
               <menuitem
   label="Save"/>
               <menuseparator/>
               <menuitem
   label="Exit"/>
            </menupopup>
         </menu>
   </menubar>

</toolbox></window>
```

screen. Another feature is that we can add a special grippy control so that the toolbar can be separated from the window and moved around in a separate overlayed window. We will not use this feature for our current menus – this feature is mainly used for toolbars.

We next need to create a menubar that our menu entries will sit on. This refers to the typically gray space that is behind the menu items. We can add this by using the <menubar> tag:

```
<menubar id="menubar">
```

Note how we have set an id for this menu bar. This is common practice when dealing with menus and buttons in XUL.

We are now ready to add a menu item. To do this we use the <menu> tag and pass it an id attribute and name it with the label attribute:

```
<menu id="filemenu" ➥
label="File">
```

We now need to fill the menu with a number of items. To do this we need to first add a special popup menu that can contain the menu items. We do this by using the <menupopup> tag:

```
<menupopup id="filepopup">
```

With our popup menu added, we are now ready to add some items to our menu. We do this by using the <menuitem> tag to add each item in turn:

```
<menuitem label="New"/>
<menuitem label="Open"/>
<menuitem label="Save"/>
```

Something you may have noted is that there is no id attribute for each menu. We will be using a slightly different method of handling which menu item is clicked later in the series.

When you are creating your menus, you may want to separate a single menu into different sections. We can add our own lines to the menu by using a menu separator. We simply add the <menuseparator> tag:

```
<menuseparator/>
```

Our final item can be a standard Exit option, below the separator:

```
<menuitem label="Exit"/>
```

To end, we will add all of the closing tags that make up our menu structure:

```
</menupopup>
    </menu>
  </menubar>
</toolbox>
</window>
```

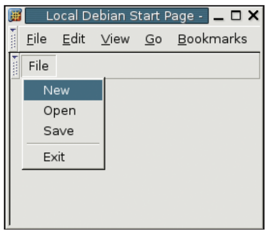

Figure 1: A simple File menu with a separator line included.

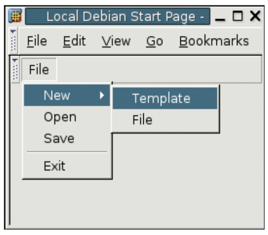

Figure 2: Adding a sub-menu to a main menu is simple in XUL.

Our completed file, so far is shown in Listing 2. You can see our completed menu structure in Figure 1, above.

Adding more menu features

One of the most common requirements in a menu structure is a sub-menu. This special type of menu appears when you hover your mouse over an existing menu entry.

To create one of these sub-menus, we need to edit some of our existing code and add another <menupopup> structure. We will add a sub-menu to our New menu item and the first line to change is:

```
<menuitem label="New"/>
```

When we add a popup sub-menu, we need to change the item that is the parent of the sub-menu from a <menuitem> to a <menu> tag:

```
<menu id="newmenu" label="New">
```

Here we have created a new menu structure, and we are now ready to add our additional popup sub-menu:

```
<menupopup id="secondpopup">
  <menuitem label="Template"/>
  <menuitem label="File"/>
</menupopup>
```

Remember to add the closing tag:

```
</menu>
```

You should now see something similar to Figure 2.

Building a complete interface

XUL shows its true power when you start tying together a number of different widgets to form a complete interface.

We are going to create a complete interface in XUL that can be used as a foundation for your own XUL interfaces. We will be building this interface from the ground up as an information management tool. This will involve us having different parts of the screen dedicated to different functions and uses. Although we are not going to be writing the functionality for the interface, we will use this interface as a basis for us starting to write a complete XUL application.

To make the interface easier to understand, we will go through each line of code and then present the full source listing at the end of the article.

To begin with, we will first add our boilerplate XUL file code as in Listing 1.

This code simply specifies to Mozilla that we are dealing with XUL and then uses the <window> tag to create our parent window. With this code complete we should now specify the file that will contain the Javascript that provides functionality for our interface:

```
<script src="code.js"/>
```

In our example interface here, we are not actually writing any Javascript and purely concentrating on the XUL. If you were writing a functional XUL application you would need to ensure that you specify the <script> tag as the first tag within the <window> tag block.

Our first part of the interface to create is our menu structure. We will begin by specifying a <toolbox> tag to indicate that our menu is positioned at the top of the main window. This will ensure that our menus and toolbars (we will cover these later in the series) are grouped together in the tool box:

```
<toolbox>
```

Next we can create our main menu bar and place a <menu> and <menupopup> on there. Remember that every menu is specified with a <menu> tag (this tag actually adds the menu name to the menubar) but you also need to create the actual popup menu (the menu that appears when you click on the menu item. Here is the code:

```
<menubar id="samplemenubar">
  <menu id="filemenu" ↩
label="File">
    <menupopup id="filepopup">
```

At this point we have no items in our popup menu. Our first item, New, is a little different because it is itself a popup menu. To add this sub-menu we create our <menu> and <menupopup> tags, and then we add our <menuitem> tags. It is important to note that although the other entries on our File menu are added with <menuitem> tags, the New entry is added with a <menu> tag as it is a sub-menu:

```
<menu id="newmenu" label="New">
  <menupopup id="secondpopup">
    <menuitem label="Template"/>
    <menuitem label="File"/>
  </menupopup>
</menu>
```

With the New menu and submenu complete, we can now add the rest of our items to the File menu. We do this with a number of <menuitem> entries, and we use a <menuseparator> tag to create a line between the menu items:

```
<menuitem label="Open"/>
    <menuitem label="Save"/>
    <menuseparator/>
    <menuitem label="Exit"/>
  </menupopup>
</menu>
```

We now add a second menu to the toolbar. This will be the Edit menu. Again we use the <menu>, <menupopup> and <menuitem> tags:

```
<menu id="editmenu" ↩
label="Edit">
  <menupopup id="editpopup">
```

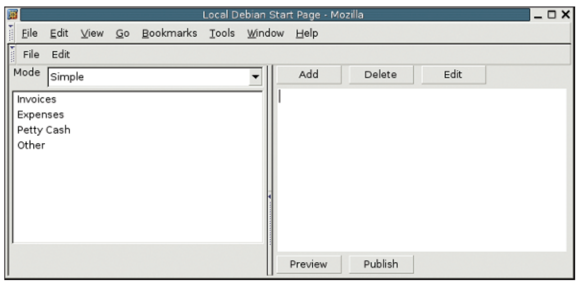
Local Debian Start Page - Mozilla
File Edit View Go Bookmarks Tools Window Help
File Edit
Mode
Simple
Add
Delete
Edit
Invoices
Expenses
Petty Cash
Other
Preview
Publish

Figure 3: A complete XUL interface.

```
    <menuitem label="Undo"/>
    <menuitem label="Redo"/>
  </menupopup>
</menu>
```

To finish our menu structure we now need to add the <menubar> and <toolbox> closing tags:

```
</menubar>
</toolbox>
```

The main part of our interface is going to comprise of a left pane with a combo box and list box, a resizable divider bar, and on the right we will have a main text entry area and some other buttons. With this in mind, we will need to have some widgets on top of each other (the combo box is on top of the list box for example) and we will have some widgets sitting next to each other (e.g. buttons). This will involve combining the <hbox> and <vbox> tags to group these widgets together in particular ways.

We will begin by first using the <box> tag to indicate that we are using a box layout for all of our widgets and then then we are going to create a <vbox> tag which will stack our combo box and list box on top of each other. Each of the widgets uses flex = "1" to stretch and take up all the space; we use flex across our full source code:

```
<box flex="1">
<vbox flex="1">
```

Our first widgets are going to be a label and a combo box, and we will encase these widgets in <hbox> tags so that they are positioned side by side. We first create a <label> tag with the text 'Mode' and then we create a combo box (this is referred to in XUL as a menu list) with the <menulist> tag. Inside this tag we create a number of <menuitem> entries for each item in the box. We have also set the editable attribute to false for our menu list so that users cannot change the contents of the box:

```
<hbox>
<label value="Mode"/>
<menulist
editable="false"
   flex="1">
  <menupopup>
    <menuitem label=
     "Simple"/>
    <menuitem label="
     Advanced"/>
    <menuitem label=
     "Expert"/>
  </menupopup>
</menulist>
</hbox>
```

The widget to add is the list box. To create this we use the <listbox> tag and add each item with the <listitem> tag:

```
<listbox>
  <listitem label="Invoices"/>
  <listitem label="Expenses"/>
  <listitem label="Petty Cash"/>
  <listitem label="Other"/>
</listbox>
```

Earlier in the code we opened a <vbox> tag to stack our combo box and list box on top of each other. We will need to now close this with our closing <vbox> tag:

```
</vbox>
```

So far, we have created our widgets that appear on the left side of the interface. In cases where you are clearly separating one side of an interface from the other, it is a good idea to use a <splitter> tag to provide a resizable divider between the two sides. One of the features of a splitter is that we can collapse one side, and we use the 'collapse' attribute to specify this as the widgets before the splitter in the code (our combo box and list box). We have also added a <grippy> tag to provide a handle to move the splitter:

```
<splitter collapse="before">
  <grippy/>
</splitter>
```

For the widgets on the right of the splitter, we are going to stack groups of widgets on top of each other. We will first create a <vbox> tag:

```
<vbox flex="1">
```

Our first block of widgets is three buttons that will appear near the top of the window. We create a <hbox> tag to place each side by side, and then add each with the <button> tag:

```
<hbox>
<button id="addbutton" ↩
label="Add"/>
<button id="deletebutton" ↩
label="Delete"/>
<button id="editbutton" ↩
label="Edit"/>
</hbox>
```

The next widget is a large text entry widget. This is the kind of widget that would be used to enter text in a text editor, so we need to make sure that the widget can handle more than one line of text. We can set this with the multiline attribute:

```
<textbox id="posttext" ↩
multiline="true" flex="1"/>
```

Our final set of widgets is a collection of buttons that work in exactly the same way as the Add, Delete and Edit buttons that we created earlier:

```
<hbox>
<button id="findbutton" label=↩
"Preview" default="true"/>
<button id="cancelbutton" ↩
label="Publish"/>
</hbox>
```

To complete our code, we now add the <vbox>, <box> and <window> closing tags:

```
</vbox>
</box>
</window>
```

Listing 3 is a complete listing of the code. When you run this code, you should see something similar to Figure 3.

Moving forward

In this issue we have brought together the summary of our XUL interface building knowledge. As we continue to explore XUL, we are going to explore the functionality of a XUL application in more detail, and this will also include us using PHP to make more dynamic XUL files.

Until we resume next month, keep playing with the different XUL interface tags, and experimenting with different interfaces and combinations of widgets – the more you experiment with XUL, the easier you will be able to use it to solve your problems. ■

Listing 3: Complete Listing of third.xul

```
<?xml version="1.0"?>
<?xml-stylesheet
href="chrome://global/skin/"
type="text/css"?>

<window
    id="testwindow"
    title="XUL Interface"

xmlns="http://www.mozilla.org/
keymaster/gatekeeper/there.is.
only.xul">

<script src="code.js"/>

<toolbox>
  <menubar id="samplemenubar">
      <menu id="filemenu"
label="File">
                <menupopup
id="filepopup">
            <menu id="newmenu"
label="New">

<menupopup id="secondpopup">

<menuitem label="Template"/>

<menuitem label="File"/>

</menupopup>

</menu>
            <menuitem
label="Open"/>
            <menuitem
label="Save"/>
            <menuseparator/>
            <menuitem
label="Exit"/>
         </menupopup>
        </menu>
     <menu id="editmenu"
label="Edit">
        <menupopup
id="editpopup">
          <menuitem
label="Undo"/>
                  <menuitem
label="Redo"/>
                <menupopup>
     </menu>
  </menubar>
</toolbox>
<box flex="1">
<vbox flex="1">
<hbox>
<label value="Mode"/>
<menulist editable="false"
flex="1">
  <menupopup>
    <menuitem label="Simple"/>
    <menuitem
label="Advanced"/>
    <menuitem label="Expert"/>
  </menupopup>
</menulist>
</hbox>
<listbox>
  <listitem label="Invoices"/>
  <listitem label="Expenses"/>
  <listitem label="Petty
Cash"/>
  <listitem label="Other"/>
</listbox>
</vbox>
<splitter collapse="before">
  <grippy/>
</splitter>
<vbox flex="1">
<hbox>

<button id="addbutton"
label="Add"/>
<button id="deletebutton"
label="Delete"/>
<button id="editbutton"
label="Edit"/>
</hbox>
<textbox id="posttext"
multiline="true" flex="1"/>
<hbox>
<button id="findbutton"
label="Preview"
default="true"/>
<button id="cancelbutton"
label="Publish"/>
</hbox>
</vbox>
</box>
</window>
```