

Mozilla XUL: Diseño de Interfaces.

Jugando con Bloques

En el primer capítulo de nuestro tutorial de programación en XUL del último mes exploramos algunos de los diferentes componentes y facilidades que XUL ofrece a la hora de construir interfaces gráficas. Si bien era bonita de ver, nuestro interfaz era lo más inconsistente que podamos imaginar. - muy bonita, pero sin materia gris. **POR JONO BACON**

Este mes vamos a incrementar nuestros conocimientos en construcción de elementos de interfaces y añadirles funcionalidades reales y útiles que se utilizarán para crear los bloques de construcción de aplicaciones futuras usando XUL.

Bloques de Interactivos

Cualquier tipo de herramienta gráfica, donde se incluye XUL, es inútil si no dispone de funciones que traten la interacción. Cuando un usuario pulsa un botón, desplaza una barra de desplazamiento, selecciona una pestaña, selecciona un objeto de un menú o pulsa en el botón de una barra de herramientas, esperamos que suceda algo. Por ejemplo,

Otros lenguajes y XUL

Una de las mayores preguntas que se realiza al mirar a XUL es si hay otros lenguajes disponibles para escribir código XUL. Actualmente el único lenguaje soportado es Javascript, si bien se está pensando en la inclusión de otros lenguajes en versiones futuras de Mozilla. Entre estos lenguajes puede que estén incluidos Python o C#. Muchas de las discusiones entre los desarrolladores se giran alrededor de la creación de un nuevo nivel de funcionalidad de la aplicación Mozilla: la máquina virtual de Mozilla. Podemos encontrar más información al respecto del progreso del proyecto Mozilla en las Webs Mozillazine (<http://www.mozillazine.org/>) y en Planet Mozilla (<http://planet.mozilla.org/>).

si pulsamos en el menú Archivo y luego en el Salir, esperamos que la aplicación se cierre como se supone que ha de hacer. Este tipo de interacción involucra 2 tipos de procesos: un *Manipulador de Eventos* y una *Función*.

Algunos lectores puede que estén familiarizados con este tipo de funcionalidad por otras herramientas gráficas: Qt tiene sus Señales/Slots, por ejemplo. La premisa básica es que cada tipo de control gráfico (como un botón, barra de desplazamiento, menú, barra de herramientas, caja de texto, ...) tenga diferentes formas en las que el usuario pueda interactuar con él. Por ejemplo, un botón dibujado en la pantalla podemos pulsarlo, con una lista de objetos en una caja podemos seleccionar un componente y con una barra de desplazamiento podemos desplazarlo. Cada uno de estos diferentes tipos de interacción se llama *Evento*. Algunos componentes o widgets (los controles gráficos son normalmente llamados en inglés widgets) también tienen diferentes formas de eventos para diferentes formas de interacción.

Cada uno de estos eventos no es útil si no podemos responder a ese evento con algo útil, y para hacer esto necesitamos usar una función. Una función es simplemente un trozo especial de código que podemos escribir y que hace algo útil. Esto puede ser cambiar el texto de algunas partes de la interfaz, añadir algo de información al componente, procesar información o cualquier otra cosa que sea de utilidad.



Los beneficios de usar funciones es que pueden ser utilizadas una y otra vez por diferentes eventos. Por ejemplo, si escribimos una función que crea un documento nuevo queremos conectarla al objeto del menú *Archivo > Nuevo* y también al botón de la barra de herramientas utilizado para abrir un nuevo documento. De esta forma todas las cosas se conectan entre sí como una sola.

Código Bajo Control

Con toda esta teoría discutida, probablemente el lector se estará preguntando como encaja todo esto en el contexto general. ¿Cómo creamos un manipulador de eventos, cómo creamos una función y, sobre todo, cómo agrupamos todo junto para qué funcione de forma conjunta? La respuesta a todas estas preguntas es Javascript.

Antes de que continuemos debemos hablar un poco de Javascript. Muchas personas se mofan de solo pensar en Javascript y lo consideran solo útil para crear esos molestos mensaje que reptan por la barra de estado de los navegadores web y para crear menús que pueden hacer una página Web aparentar ser incómoda y que solo funcionan parcialmente en algunos visores.

Sí es cierto, Javascript puede ser utilizado para todas esas cosas y es posiblemente la cara más pública del lenguaje. Realmente pienso que Javascript es un lenguaje maravilloso, compacto y útil que puede ser considerado como el pegamento de los

Listado 1: first.xul

```

01 <?xml version="1.0"?>
02 <?xml-stylesheet
03 href="chrome://global/skin/"
04 type="text/css"?>
05
06 <window
07 id="test-window"
08 title="Test Program"
09
10 xmlns="http://www.mozilla.org/
11 keymaster/gatekeeper/there.is.
12 only.xul">
13
14 <button id="name"
15 label="Mi botón"/>
16
17 </window>

```

visores Web: es muy útil para unir diferentes tipos de funcionalidades. En el terreno de XUL usamos un objeto especial llamado Documento Objeto Modelo (DOM), la idea tras DOM es que los elementos en una página Web deben ser todos accesibles en código de forma que todos podemos ver y cambiar los contenidos de estos cuando y como necesitemos. Por ejemplo sería útil cambiar el título, la fecha y etiquetas de los botones de una página Web cuando interactuemos con ella.

Listado 2: Código Completo

```

01 01 <?xml version="1.0"?>
02 <?xml-stylesheet
03 href="chrome://global/skin/"
04 type="text/css"?>
05
06 <window
07 id="test-window"
08 title="Programa de Prueba"
09
10 xmlns="http://www.mozilla.org/
11 keymaster/gatekeeper/there.is.
12 only.xul">
13
14 <script
15 src="functions.js"/>
16
17 <button id="name"
18 label="Mi botón"
19 onclick="func_showdialog();"/>
20
21 </window>

```

Un ejemplo de este tipo de funcionalidad con el que todos estamos familiarizados es la del software de foros web modernos. Cuando usamos los botones de formato al componer un mensaje para el foro, el botón añadirá etiquetas de formato a la caja de texto principal. ¿Cómo sabe comunicarse el botón con la caja de texto del mensaje cuando lo pulsamos? Sencillo: usa DOM.

DOM funciona de forma que crea un árbol con todos los elementos en nuestra página Web. Veamos el siguiente código como un ejemplo:

```

01 <html>
02 <head>
03 <title>Mi
04 Sitio Web</title>
05 </head>
06 <body>
07 <div>
08 <h1>Bienvenidos a
09 mi sitio web</h1>
10 </div>
11 <div>
12 Contenido.
13 </div>
14 </body>
15 </html>

```

Cada una de las etiquetas del código anida dentro de otra, menos la etiqueta

<html> que es la etiqueta padre. La etiqueta padre es como la raíz del árbol y cada una de las otras etiquetas son como ramas que surgen del árbol. Por ejemplo, si miramos a la etiqueta <body>, tenemos dos etiquetas <div> que están dentro de esta. Estas dos etiquetas <div> son como dos ramas. Debajo podemos ver como nuestras etiquetas forma un árbol con la etiquetas <html> como raíz:

```

<html>
  <head>
    <title>Mi Sitio Web</title>
  <body>
    <div>
      <h1>Bienvenidos a
      mi sitio web</h1>
    <div>
      Contenido.

```

Con nuestras etiquetas en un árbol como este podemos usar DOM para encontrar ciertas etiquetas y mostrarlas o conseguir la información requerida de ellas. De esta forma podemos encontrar una etiqueta en particular y modificarla de cualquier forma. Como resultado, DOM no sólo nos provee de un método para encontrar las etiquetas adecuadas, si no que también nos proporciona un método para actualizarlas.

DOM y XUL

Para empezar de forma rápida utilizaremos como ejemplo simple la creación de un botón con un sistema de evento/funcionalidad. Lo primero de todo será crear un nuevo archivo llamado *first.xul*, al que añadiremos el siguiente código (ver listado 1 en la página previa).

Cuando cargamos este archivo en un visor Mozilla obtendremos un simple botón como esperamos. Nuestro primer paso ahora es añadir un manipulador de eventos que especifique a que tipo de interacción queremos responder. Cada evento se activa por un manipulador especial que especifica con que tipo de interacción estamos tratando. La tabla de eventos XUL es una lista de eventos soportado concurrentemente que está disponibles en XUL (lista conseguida de *XULPlanet.com*).

El Señor de los Botones

Empezaremos explorando uno de los manipuladores de eventos más común:

Listado 3: second.xul

```

01 01 <?xml version="1.0"?>
02 <?xml-stylesheet
03 href="chrome://global/skin/"
04 type="text/css"?>
05
06 <window
07 id="test-window"
08 title="Programa de Prueba"
09
10 xmlns="http://www.mozilla.org/
11 keymaster/gatekeeper/there.is.
12 only.xul">
13
14 <script src="second.js"/>
15
16 <textbox id="textbox"/>
17 <button id="showinfo"
18 label="Mostrar Información"
19 onclick="func_getinfo();"/>
20
21 </window>

```

onclick. Para añadir este manipulador necesitamos añadir *onclick* a la línea de código que crea el botón y luego especificar que debe ocurrir. Cambiemos el código del botón a esta línea:

```
<button id="name" label="Mi botón" onclick="alert('Hola');"/>
```

En este código añadimos el controlador *onclick* y dentro de las comillas decimos lo que queremos que haga el controlador. En este simple ejemplo usamos la función de alerta de Javascript para que aparezca una caja de dialogo con la palabra “Hola” dentro. Si ahora salvamos el código, recargamos la página y pulsamos el botón veremos las cajas que aparecen cuando pulsemos con el ratón.

Todo Funciona

Como podemos observar en nuestro primer ejemplo, añadir funcionalidad a un widget es muy sencillo. Al margen de la sencillez, nos encontramos con el problema de que nuestro código va a ser muy largo y confuso si ponemos toda su funcionalidad entre marcas de comillas. ¿Qué hacemos si lo que necesitamos es hacer algo de matemáticas, procesado expansivo y magia con la programación que requeriría mucho código escrito? La respuesta es hacer nuestra propia función.

Una función es un bloque de código que puede ser llamado por otro trozo de código. Podemos considerar las funciones como libros dentro de librerías. Podemos estar leyendo un libro y no entender un asunto concreto, cogiendo entonces otro libro que nos aclare el asunto. Lo que nos permite básicamente una función es empaquetar conjuntamente en código un complejo paquete de programación y acceder a ésta de una forma simple en sólo una línea. Si bien las funciones pueden resultar complejas

y gestionar muchas funcionalidades, también pueden ser simples. Exploraremos nuestra primeras funciones en Javascript con un ejemplo muy sencillo. Para usar funciones se recomienda poner todas las funciones en un archivo separado y acceder al mismo desde nuestro archivo de comandos XUL. Para empezar debemos crear un archivo llamado *functions.js* y añadirle el siguiente código:

```
function func_showdialog()
{
    alert("Esto es una función");
}
```

Este código contiene una serie de elementos claves. La primera línea dice que

estamos creando una función (“function”) y que queremos llamarla *func_showdialog*. Los paréntesis vacíos se usan para indicar que nuestra función no procesa nada. Una actividad propia de las funciones es pasarles información entre paréntesis, procesando la información y devolviendo el resultado. Con nuestra función simplemente estamos ejecutando el código dentro de ella y no procesado nada, siendo este el motivo de que los paréntesis estén vacíos.

Las llaves indican la extensión de la función. El código dentro de la función comienza tras el símbolo { y termina tras el símbolo }. En nuestra función actual solo tenemos una línea que nos da un caja de dialogo de alerta familiar. El

Eventos XUL

Evento	Descripción
onclick	se llama a la acción de pulsar y soltar el ratón sobre un elemento. Sólo debemos usar el evento onclick cuando tenemos una razón para responder a pulsaciones de ratón. Para botones, elementos de menús y similares debemos usar <i>oncommand</i> como respuesta porque responde también a usuarios que usan el teclado u otros dispositivos.
onmousedown	acción de pulsar el ratón sobre un elemento. El gestor de eventos será llamado tan pronto el botón del ratón sea presionado, incluso antes de ser soltado.
onmouseup	cuando el botón del ratón se suelta estando sobre un elemento.
onmouseover	se llama a la acción de mover el puntero del ratón sobre un elemento. Podemos usar esta acción para destacar un elemento, si bien CSS proporciona una manera de hacer esto automáticamente, por lo que no debemos usar un evento para esto. Pero por ejemplo puede que queramos mostrar un texto de ayuda sobre una barra de menú.
onmousemove	acción de mover el puntero del ratón mientras está sobre un elemento. El evento será ejecutado muchas veces si el usuario mueve el ratón, por lo que debemos evitar usar este evento en la medida de lo posible.
onmouseout	se llama a la acción de mover el puntero del ratón fuera de un elemento. Puede que entonces dejemos de destacar el elemento o quitemos el texto marcado.
oncommand	evento que ocurre cuando se selecciona un botón o elemento de un menú. Para menús debemos añadir este evento al elemento <i>menuItem</i> . Debemos usar este evento en lugar de manejar el ratón nosotros mismos puesto que el usuario puede seleccionar el botón o el elemento del menú usando el ratón, una tecla de acceso directo o atajos del teclado.
onkeypress	ocurre cuando una tecla se presiona y suelta cuando un elemento tiene el enfoque. Puede que queramos usar este evento para añadir una tecla de acceso directo extra o para verificar la posibilidad de incluir determinados caracteres en un campo. Veremos como crear accesos directos desde el teclado en posteriores números.
onkeydown	se ejecuta cuando se presiona una tecla mientras un elemento tiene el enfoque. Debemos observar que el evento se ejecutará en cuanto se presiona la tecla, incluso antes de que ésta sea soltada. Puede que no usemos este evento habitualmente, puesto que hay otros eventos de teclado más usuales.
onkeyup	se ejecuta cuando una tecla es soltada teniendo un elemento el enfoque.
onfocus	se ejecuta cuando un elemento obtiene el enfoque bien mediante la pulsación del ratón o por el uso del tabulador. Podemos usar este evento para destacar un elemento o mostrar texto.
onblur	se ejecuta cuando un elemento deja de tener el enfoque debido a que hemos pulsado otro elemento o pulsamos el tabulador. Podemos usarlo para verificar información o cerrar elementos emergentes. No obstante es recomendable verificar campos al pulsar el botón de confirmación (OK).
onload	se ejecuta en una ventana al abrirse. Normalmente añadiremos este evento a la etiqueta de una ventana para inicializarla. Estos añadirá usualmente valores por defecto en condiciones incluidas en el código.
onunload	se ejecuta cuando se cierra una ventana. Normalmente añadiremos esta etiqueta de ventana para registrar información antes de que esta se cierre.

Listado 4: *second.js*

```
01 function func_getinfo()
02 {
03     var
04     info=document.getElementById('
05     textbox');
06     alert(info.value);
07 }
```

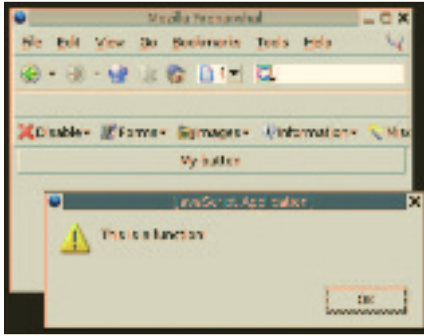


Figura 1: Nuestro primer código Javascript habilitado.

punto y coma que aparece tras cada una de las líneas de la función es importante. Se utiliza para indicar el final de cada línea en Javascript.

Conexión

Con nuestra función acabada ya estamos listos para conectarla a nuestro código XUL principal. primero necesitamos cargar el archivo *funcions.js* en nuestro archivo XUL como fuente de nuestro código. Para hacer esto usamos la etiqueta `<script>` dentro de la etiqueta `<window>` para especificar la fuente del archivo. Adicionalmente también necesitamos cambiar el contenido del atributo `onlick` del botón al nombre de nuestra función. Para clarificar estas modificaciones mostramos el código completo en el listado 2.

La principal razón por la que he reproducido el código complete ha sido para demostrar el correcto emplazamiento del código nuevo. Si pusiésemos la etiqueta `<script>` debajo de las etiquetas `<?xml>` (un error muy común en aquellos que piensan que el script se aplica al archivo entero) obtendríamos un error y nuestro esfuerzos serían baldíos. Debemos estar seguros de que nuestro script está dentro de la etiqueta `<window>`.

Exploración de DOM

Avancemos y exploremos DOM un poco más. Para empezar, crearemos un poco de código nuevo. Este nuevo archivo lo vamos a llamar *second.xul*, al que le añadiremos el código del listado 3.

En este código tenemos dos elementos gráficos principales: una caja de texto y un botón. Es importante observar como cada widget tiene un atributo `id` que le proporciona un nombre único.

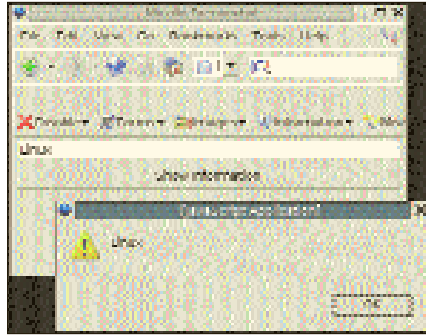


Figura 2: Usando DOM para conectar nuestras widgets.

Debemos asegurarnos de que este atributo es único y fácil de recordar, puesto que nos referiremos a los widgets por él. En nuestro código también hemos cambiado el nombre de la función a `fun_getinfo()` al igual que el nombre del archivo de nuestra función en la línea 9 a *seconds.js*.

El siguiente archivo que debemos editar es *seconds.js*. Añadimos el código del listado 4. En este código es donde realmente ocurre la acción. El objetivo de este código es que el usuario pueda escribir algo en la caja de texto y que los contenidos de la caja sean mostrados en la caja de alertas de diálogo. Para hacer esto necesitamos comunicarnos con la caja de texto, obtener la información y ponerla en la caja de alertas.

En la primera línea del código de la función creamos una nueva variable llamada `info`. Para hacer esto usamos la palabra clave `var` para indicar que `info` es una variable. Para aquellos no familiarizados con variables, una variable nos permite almacenar información en la memoria del ordenador y referirnos a ella con un nombre. Podemos pensar en las variables como una caja de cartón con un nombre escrito en su lateral. Si metemos algo en la caja nos podemos referir a ella por el nombre que aparece en el lateral. En este caso el nombre escrito en el lateral es `info`.

En la misma línea de nuestro código `var info` fijamos los contenidos de la variables con el código a la derecha del signo `=`. Este es el código que realmente coge la información de la caja de texto. Dentro del código utilizamos la función `getElementById` de Javascript para acceder al widget con la identidad

(ID) `textbox` que es nuestra caja de texto.

Podemos observar que la función `getElementById` está escrita justo al lado de la palabra `document`. La forma de funcionar es que `document` se refiere a nuestro documento XUL principal y a los widgets incluidos en él.

Punto

El símbolo del punto (.) indica que el código a la derecha del mismo debe ser aplicado al código a la izquierda del símbolo. En este caso buscamos un widget dentro del id `textbox` dentro de nuestro documento principal. Con esta línea completa tenemos los contenidos de nuestra caja de texto almacenados en nuestra variable `info`. El concepto de usar puntos para indicar dónde aplicar la función es común a muchos lenguajes de programación. Usamos este concepto de nuevo en la segunda línea al usar la función `value` para obtener el valor de la variable `info`. Anidamos este código dentro de la función alerta para mostrarlo en la caja de diálogo. Debemos prestar atención a que no se usan comillas dobles en el código de la caja de alertas. Solo usamos comillas dobles cuando queremos imprimir texto (llamado *string* o *cadena* en castellano) en la caja de alertas. En nuestra caja de alertas, en esta ocasión, estamos imprimiendo los contenidos de la variable.

Conclusión

En este número hemos realizado unos importantes primeros pasos hacia conseguir un interfaz funcional XUL. Todavía nos falta camino por andar, pero estos cimientos harán nuestros programas futuros sean más sencillos de escribir y entender. Siempre es importante tener claro los fundamentos antes de continuar el camino. En el siguiente capítulo vamos a continuar con algunas funciones y funcionalidades más elaboradas para unir diferentes partes de nuestras interfaces XUL. Hasta entonces debemos mirar con detenimiento los ejemplos que hemos visto e intentar entenderlos con el mayor detalle posible. Si hay conceptos aún confusos no debemos preocuparnos, puesto que la mayoría serán explicados con mayor detalle en próximos números. ¡Buena suerte!