Curso de Ext JS

CAP.1. Conceptos fundamentales

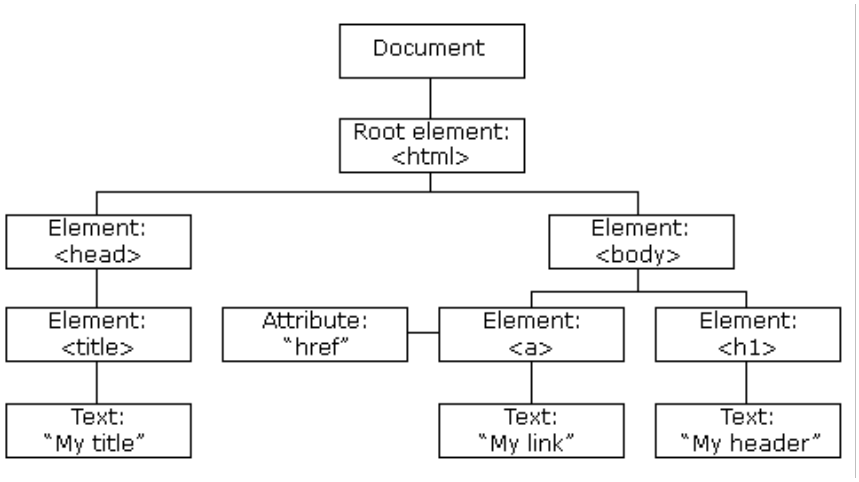
1.1 DOM

El DOM Document Object Model, es una plataforma neutral que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de los documentos. Esta estructura de objetos es generada por el navegador cuando se carga un documento.

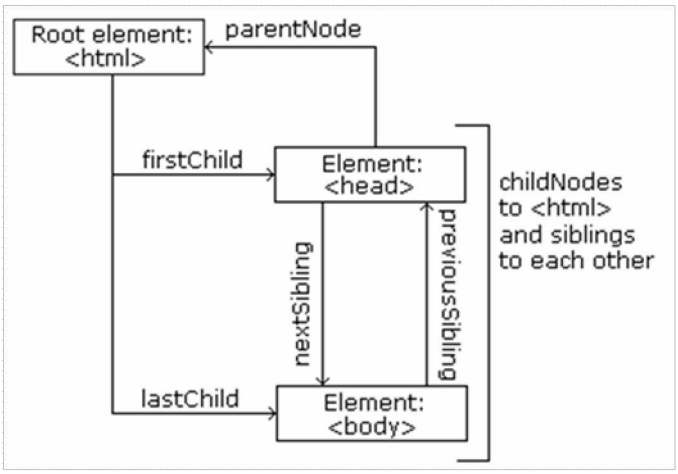
Desde el punto de vista del desarrollo de páginas web, el DOM nos indica cuál es la naturaleza de la estructura de los documentos y nos provee de una serie de recursos para poder acceder a todos sus elementos, a la relación que existe entre ellos, a sus características, sus modos de representación y los eventos que soportan.

El DOM tiene varios elementos, cada uno se encuentra dentro de una jerarquía, tanto HTML como XML tienen como base una estructura dentro de la cual se definen los nodos que pueden ser de información en el caso del XML y de objetos en el caso de HTML.

Estructura HTML:



Relación entre nodos:



Ejemplo: En el caso de un input, se puede acceder a sus propiedades o características navegando por su jerarquía, a partir de document > forms[0]:

```
<input type="text" name="caja" />
```

Para acceder al valor del input:

```
document.forms[0].caja.value
```

1.2 DHTML

El HTML Dinámico (DHTML), no es más que una forma de aportar interactividad a las páginas web, que tiene la ventaja de poder crear efectos que requieren poco ancho de banda a la hora de ejecutarse y son estos efectos los que aumentan la funcionalidad a la página, que con solo HTML simple sería imposible de realizar. Aunque muchas de las características del DHTML se podrían duplicar con otras herramientas como Java o Flash, el DHTML ofrece la ventaja de que no requiere ningún tipo de plug-in para poder utilizarlo.

Aunque las tecnologías en las que se basa el DHTML (HTML, CSS, JavaScript) están estandarizadas, la forma en que se implementan en los varios navegadores, difiere entre sí.

Por este motivo, la creación de páginas web que usen esta tecnología, puede llegar a convertirse en una tarea muy compleja, puesto que hay que conseguir que la página se visualice perfectamente en todos los navegadores.

1.3 CSS

Es un mecanismo simple para añadir estilos (fonts, colors, spacing) a los documentos Web.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

Ejemplo:

```
position               : absolute;
left                   : 50px;
top                    : 100px;
width                  : 200px;
height                 : 100px;
clip                   : rect(0px 200px 100px 0px);
visiblity              : visible;
z-index                : 1;
background-color       : #FF0000;
layer-background-color : #FF0000;
background-image       : URL(icono.gif);
layer-background-image : URL(icono.gif);
```

1.4 JavaScript

JavaScript es un lenguaje de programación interpretado (scripting), es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java.

No es un lenguaje de Programación Orientada a Objetos propiamente dicho como Java, pero gracias a que es basado en prototipos, se puede aplicar conceptos de programación orientada a objetos.

Ejemplo:

```
<script type="text/javascript">
   function fEnviaAlerta()
   {
      var texto = document.forms[0].caja.value;
      alert(texto);
   }
</script>
```

1.5 AJAX

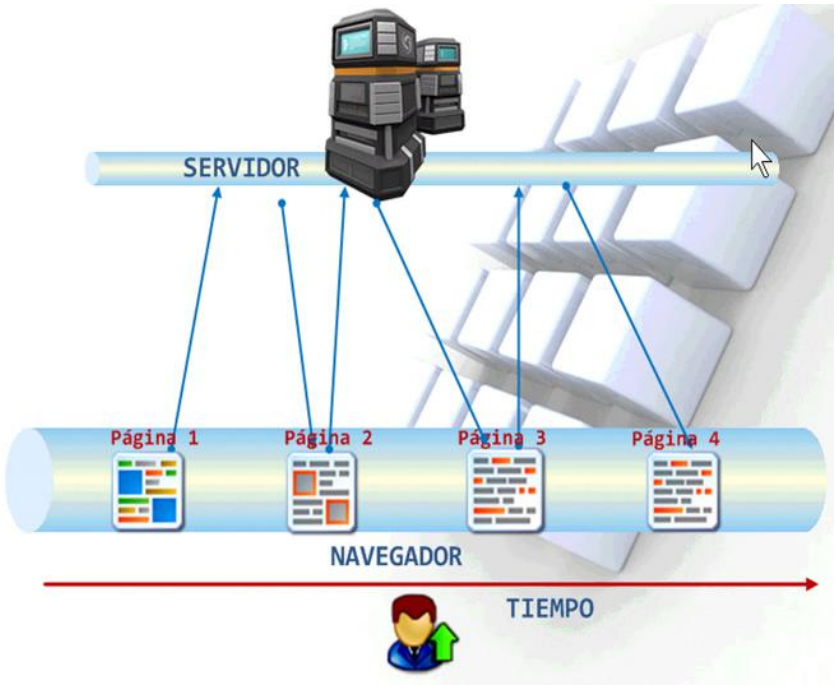
Ajax, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano.

De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

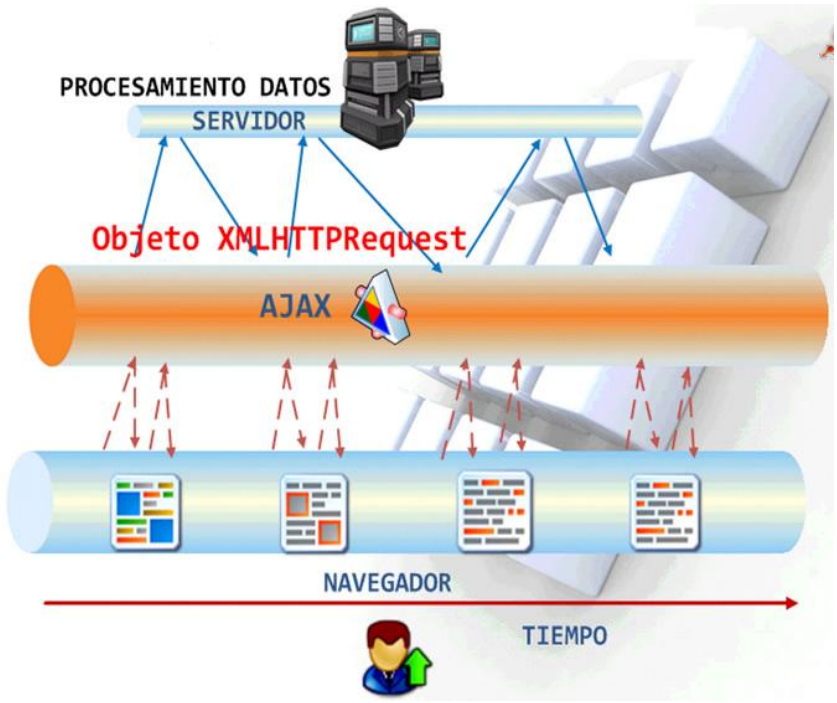
Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

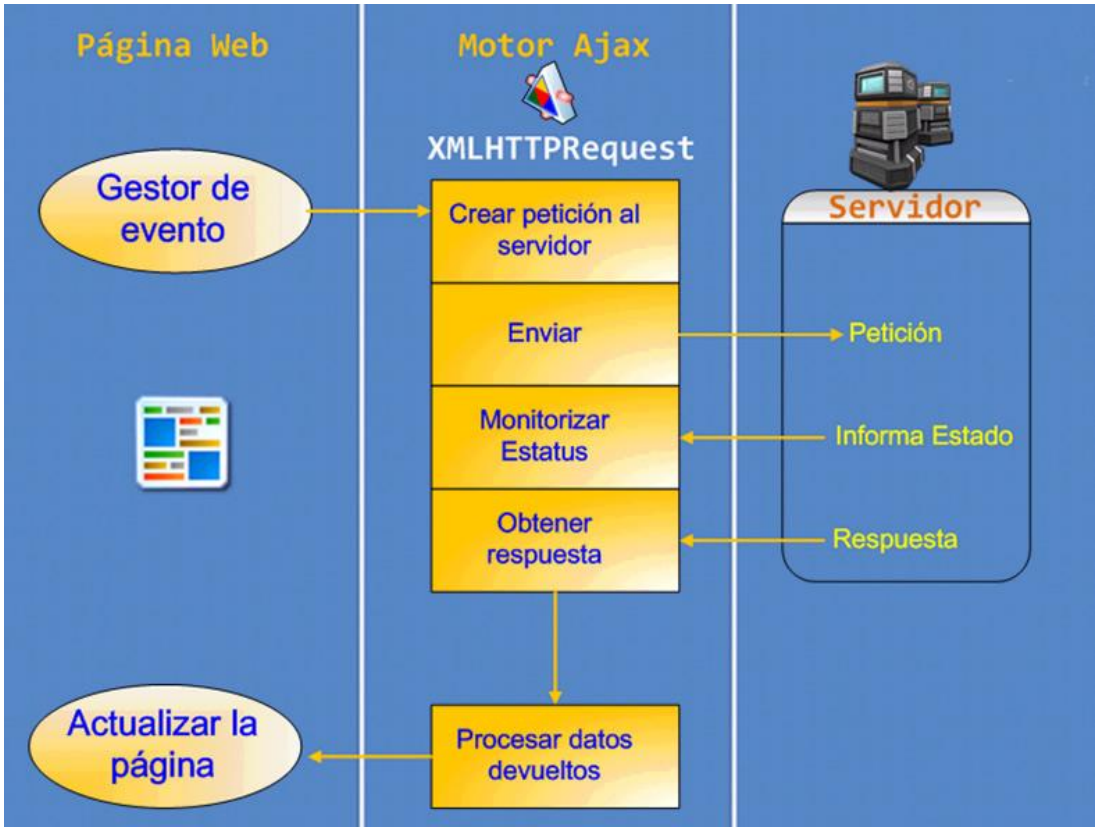
La conversación tradicional entre el cliente y el servidor es en serie, se carga una sola página a la vez, como se explica en la siguiente imagen:



La interacción AJAX entre el cliente y el servidor es asíncrona, es decir dentro de la misma página se realiza la solicitud y entrega de respuesta, gracias al objeto XMLHttpRequest, como se muestra en la siguiente imagen:



El proceso de una aplicación AJAX ser vería de la siguiente forma, como muestra la imagen:



1.5.1 El objeto XMLHttpRequest

Su objetivo es hacer peticiones asíncronas al servidor, es la columna vertebral de todas las aplicaciones AJAX. Está admitido por todos los navegadores. Microsoft lo introdujo en IE 5 como un objeto ActiveX.

Propiedades:

Propiedades	Descripción
onreadystatechange	Determina que función será llamada cuando la propiedad readyState del objeto cambie.
readyState	Número entero que indica el status de la petición:
	0 = No iniciada
	1 = Cargando
	2 = Cargado
	3 = Interactivo
	4 = Completado
responseText	Datos devueltos por el servidor en forma de string de texto
responseXML	Datos devueltos por el servidor expresados como un objeto documento.
status	Código estatus HTTP devuelto poro el servidor:
	200 = OK (Petición correcta)
	204 = No Content (Documento sin datos)
	301 = Moved Permanently (Recurso Movido)
	401 = Not Authorized (Necesita autenticación)
	403 = Forbidden (Rechazada por servidor)
	404 = Not Found (No existe en servidor)
	408 = Request Timeout (Tiempo sobrepasado)
	500 = Server Error (Error en el servidor)

Métodos:

Propiedades	Descripción
abort()	Detiene la petición actual.
getAllResponseHeaders()	Devuelve todas las cabeceras como un string.
getResponseHeader(x)	Devuelve el valor de la cabecera x como un string.
open('method', 'URL', 'a')	Especifica el método HTTP (por ejemplo, GET o POST), la URL objetivo, y si la petición debe ser manejada asíncronamente (Si, a='True' defecto; No, a='false'.)
send(content)	Envía la petición
setRequestHeader('label' , 'value')	Configura un par parámetro y valor label=value y lo asigna a la cabecera para ser enviado con la petición.

1.6 XML

XML, siglas en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

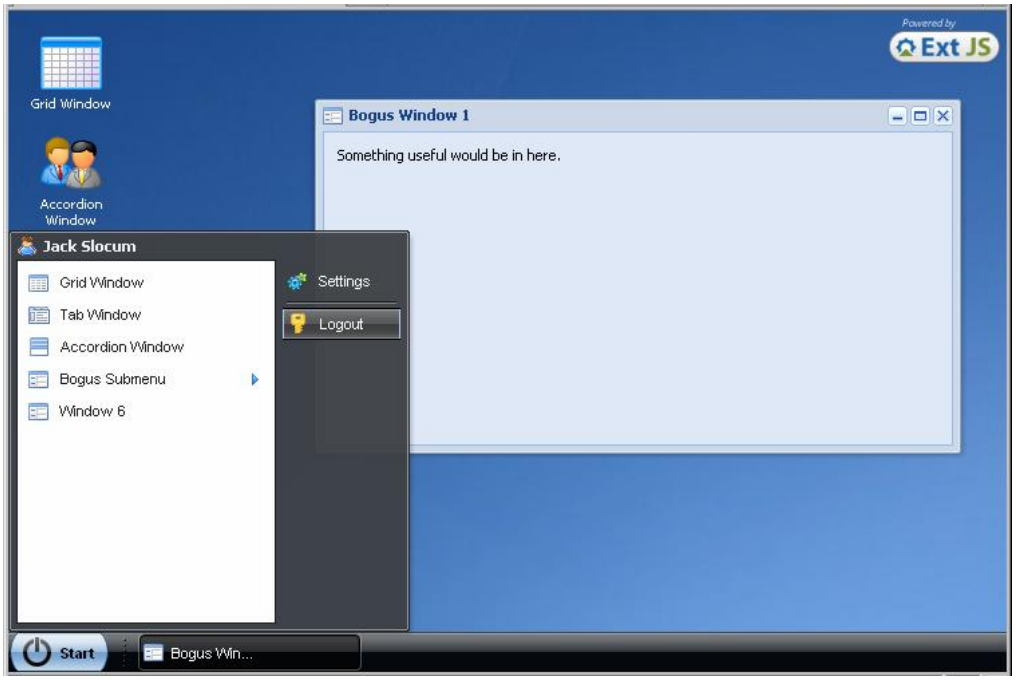
Ejemplo:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<datos>
    <total>
        125
    </total>
    <dato>
        <codigo>
            <![CDATA[77]]>
        </codigo>
        <nombre>
            <![CDATA[Aguas Termales Baños Entrada general.]]>
        </nombre>
        <datos_insumo>
            <![CDATA[ | Tipo: Entradas | Proveedores: Municipio de Baños ]]>
        </datos_insumo>
    </dato>
    <dato>
        <codigo>
            <![CDATA[78]]>
        </codigo>
        <nombre>
            <![CDATA[Aguas Termales Papallacta Entrada general]]>
        </nombre>
        <datos_insumo>
            <![CDATA[ | Tipo: Entradas | Proveedores: Termas de Papallacta ]]>
        </datos_insumo>
    </dato>
</datos>
```

1.7 RIA

Son aplicaciones web que tienen la mayoría de las características de las aplicaciones de escritorio tradicionales, estas aplicaciones utilizan un “navegador web” estandarizado para ejecutarse y por medio de un “plugin” o independientemente con una “virtual machine”, se agregan las características adicionales.
Esta surge como una combinación de las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales. Buscan mejorar la experiencia del usuario.
Normalmente en las aplicaciones Web, hay una recarga continua de páginas cada vez que el usuario pulsa sobre un enlace. De esta forma se produce un tráfico muy alto entre el cliente y el servidor, llegando muchas veces, a recargar la misma página con un mínimo cambio.
En los entornos RIA, en cambio, no se producen recargas de página, ya que desde el principio se carga toda la aplicación, y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una Base de Datos o de otros ficheros externos.

Ejemplo: Un ejemplo muy bueno de RIA es el desktop creado por el ExtJS team que utiliza un menú muy parecido al del sistema operativo Windows, tiene íconos en el escritorio y despliega ventanas que se pueden mover, cerrar y cambiar de tamaño.



1.8 JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador semántico de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando el procedimiento eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

Ejemplo:

```
(
{
  "total": "2",
  "results": [
  {
    "id_genero": "F",
    "descripcion_genero": "Femenino"
  },
  {
    "id_genero": "M",
    "descripcion_genero": "Masculino"
  }]
})
```

1.9 jQuery

jQuery es una biblioteca o framework de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2,1 permitiendo su uso en proyectos libres y privativos.2 jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Ejemplo:

```
<script type="text/javascript">
    jQuery().ready(function (){
    jQuery("#list1").jqGrid({
         url:'server.php?q=1',
         datatype: "xml",
         colNames:['Inv No','Date', 'Client', 'Amount','Tax','Total','Notes'],
         colModel:[
            {name:'id',index:'id', width:75},
            {name:'invdate',index:'invdate', width:90},
            {name:'name',index:'name', width:100},
            {name:'amount',index:'amount', width:80, align:"right"},
            {name:'tax',index:'tax', width:80, align:"right"},
            {name:'total',index:'total', width:80,align:"right"},
            {name:'note',index:'note', width:150, sortable:false}
         ],
         rowNum:10,
         autowidth: true,
         rowList:[10,20,30],
         pager: jQuery('#pager1'),
         sortname: 'id',
        viewrecords: true,
        sortorder: "desc",
        caption:"XML Example"
    }).navGrid('#pager1',{edit:false,add:false,del:false});
</script>
```

1.10 Prototype

Prototype es un framework escrito en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails.

Con la Web 2.0 las técnicas de desarrollo de páginas web necesitaban dar un gran salto. Con esto en mente nació la técnica AJAX, que gracias a Prototype permite el desarrollo ágil y sencillo de páginas Web, esto en relación al desarrollador, y provee al cliente una manera más rápida de acceder al servicio que solicita. Prototype es un Framework basado en JavasScript orientado a proporcionar al desarrollador de técnicas AJAX listas para ser usadas. El potencial de Prototype es aprovechado al máximo si se desarrolla con Ruby On Rails, esto no quiere decir que no se puede usar desde otro lenguaje, solamente que demandara un "mayor esfuerzo" en el desarrollo.

Ejemplo:

```
var nombre = $F('nameUser');
var apellido = $F('surnameUser');
var direccion = $F('directionUser');

var param = 'name=' + nombre + '&surname=' + apellido + '&direction=' + direccion;

var url = 'paginaReceptora.php';

var ajaxRequest = new Ajax.Request(url,
{
  method: 'get',
  parameters: param,
  asynchronous: true,
  onComplete: showResponse
});
```

CAP.2. Introducción a ExtJS

2.1 Un poco de historia

ExtJS fue originalmente construida como una extensión de la biblioteca YUI, en la actualidad puede usarse como extensión para las bibliotecas jQuery y Prototype. Desde la versión 1.1 puede ejecutarse como una aplicación independiente.

2.2 ¿Qué es exactamente ExtJS?

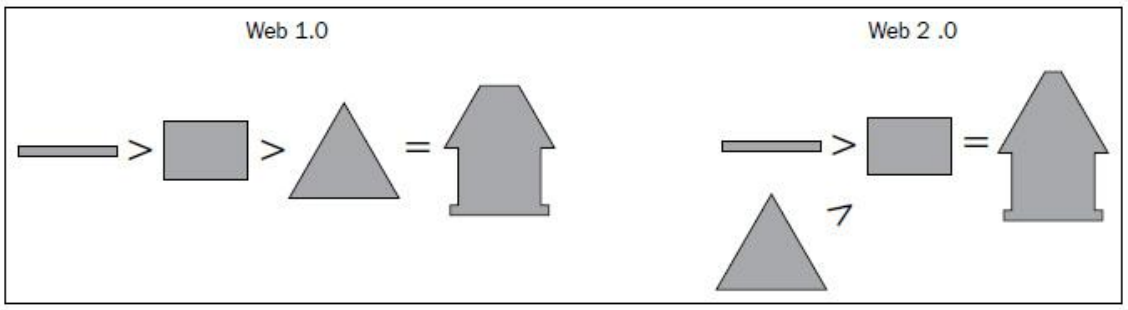
ExtJS es una biblioteca o conjunto de librerías de JavaScript para el desarrollo de aplicaciones web interactivas, usa tecnologías AJAX, DHTML y DOM.

ExtJS permite realizar completas interfaces de usuario, fáciles de usar, muy parecidas a las conocidas aplicaciones de escritorio. Esto permite a los desarrolladores web concentrarse en la funcionalidad de las aplicaciones en vez de en las advertencias técnicas.

2.3 ExtJS es asíncrono

La programación tradicional para Internet, o también llamada Web 1.0 ejecuta el código en sucesión, es decir espera a que una línea de código se complete antes de ejecutar la siguiente. Al igual que la construcción de una casa, los cimientos deben ser completados antes de que las paredes se puedan construir, a continuación, las paredes deben estar completas antes de que el techo sea construido.

Con ExtJS, que es una herramienta tipo Web 2.0, podemos fácilmente empezar por el techo de la casa antes que los cimientos estén construidos. Es algo así como imaginarse que el techo de la casa está siendo construido en una fábrica, mientras al mismo tiempo, se están construyendo los cimientos y las paredes y cuando todo esté listo, simplemente se arma todo más rápido.



Esto nos presenta otras cosas que antes no solíamos tener, ahora ya no estamos obligados a tomar el enfoque línea por línea del desarrollo web.
ExtJS nos ayuda a salir por medio de eventos y manejadores que podemos pegar a nuestra funcionalidad.

Esta manera de hacer las cosas se denomina asincronía.

CAP.3. Empezando a usar ExtJS

3.1 Descargando la librería ExtJS

- Se debe ingresar al sitio www.sencha.com.
- Escoger el framework ExtJS.
- Hacer clic en el botón Download de color verde.
- Y para el curso vamos a bajar la versión Open Source haciendo clic en el botón Download de color celeste.

3.2 ¿Dónde y cómo instalarlo?

El paquete ExtJS, no se instala, ya que es un SDK que contiene un conjunto de archivos que comprenden los recursos que ExtJS necesita para correr adecuadamente. Viene dentro de un archivo comprimido, el mismo que vamos a descomprimir dentro de la carpeta principal de nuestro servidor de páginas dinámicas favorito, en este caso lo haremos dentro de Apache>htdocs>aplicación, una vez copiado el directorio que contiene las librerías de ExtJS, cambiemos el nombre de la carpeta a "extjs" simplemente.

El SDK se compone de las siguientes carpetas y archivos importantes:

Tipo	Nombre	Descripción
Directorio	adapter	Archivos que permiten usar otras librerías dentro de ExtJS
Directorio	docs	La documentación completa de todos los componentes de ExtJS (solo corre en un servidor web)
Directorio	examples	Cientos de asombrosos y útiles ejemplos
Directorio	pkgs	Paquetes adicionales a los componentes originales que los hacen más fuertes, incluye también algunos nuevos objetos
Directorio	resources	Dependencias de las librerías de ExtJS, tales como archivos CSS e imágenes
Directorio	src	El código fuente complete de ExtJS
Archivo	ext-all.css	Es el archivo CSS principal de ExtJS, es el encargado de que los componentes se vean tan bien
Archivo	ext-all.js	Es la libería primaria de ExtJS

3.3 Documentación ExtJS

Dentro del paquete de librerías ExtJS, viene incluida la documentación, a la cual podremos acceder desde cualquier explorador web colocando la siguiente dirección local:

http://localhost/aplicacion/extjs/docs/

Así la tendremos a mano por cualquier duda que se presente.

Ejercicio: Explorar documentación

3.4 Construyendo el proyecto

Dentro de la carpeta creada en htdocs, “aplicacion”, vamos a crear las siguientes carpetas para organizar mejor nuestro proyecto:

- imagenes
- librerias
- estilos

Finalmente vamos a crear un archivo en la raíz de nuestra carpeta “aplicacion”, denominado index.html, dentro del cual vamos a llamar a las librerías que son necesarias para que incluir ExtJS en el proyecto:

```
<html>
<head>
   <title>Aplicacion</title>
   <!-- Ext Css's -->
   <link rel="stylesheet" type="text/css" href="ExtJS/resources/css/ext-all.css"/>
   <!-- Ext JS's -->
   <script type="text/javascript" src="ExtJS/adapter/ext/ext-base.js"></script>
   <script type="text/javascript" src="ExtJS/ext-all.js"></script>
   <!-- Mis JS's -->
   Espacio para colocar nuestras propias librerías
</head>
<body>
</body>
</html>
```

3.4.1 La primera librería con ExtJS

Dentro de la carpeta “librerias” colocada dentro de nuestra aplicación, vamos a crear un archivo denominado hola1.js, con el siguiente contenido:

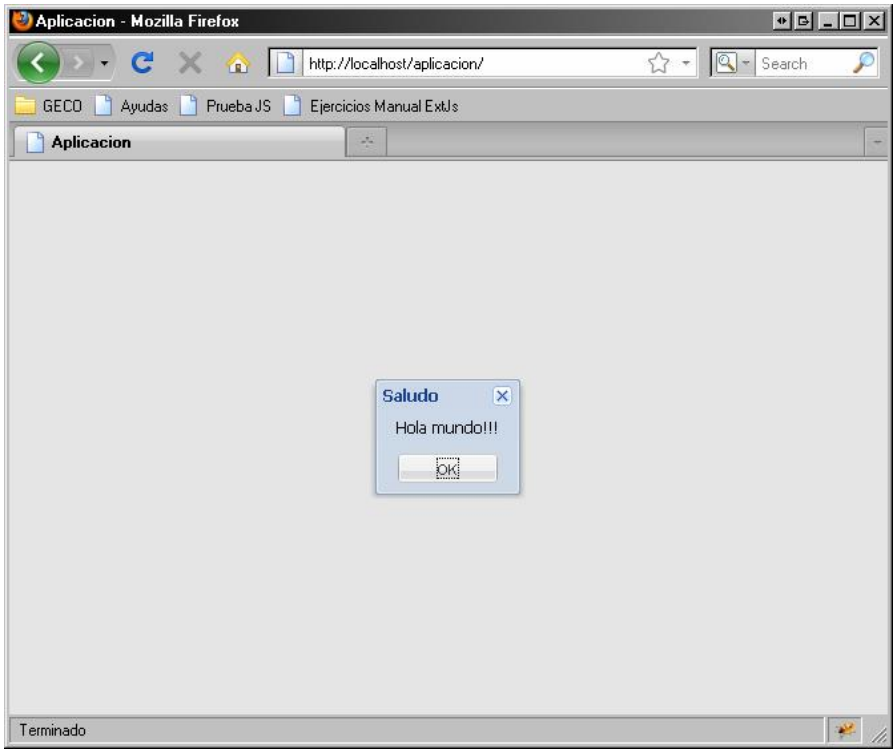
```
Ext.onReady(function()
{
  Ext.Msg.alert('Saludo', 'Hola mundo!!!');
});
```

Llamamos a la librería hola1.js dentro de index.html, en el espacio que habíamos reservado para colocar nuestras propias librerías.

Ingresamos a nuestra aplicación por medio de la siguiente dirección:

http://localhost/aplicacion/

Y esto es lo que aparece:



Analizando el ejemplo actual tenemos lo siguiente:

- **Ext.onReady:** Esta función hace que nuestro código espere antes de que el DOM esté disponible. Esto se necesita porque el JavaScript empieza ejecutando tan pronto como es encontrado en el documento, momento en el cual nuestros elementos DOM podrían no existir.
- **Ext.Msg:** Esta es la función principal usada para la creación de una alerta, ya que se ocupa de todo lo necesario para tener un cuadro de diálogo completo.

3.5 Spacer image

Antes de continuar, debemos proveer a ExtJS con algo que necesita, una imagen, que se ha denominado spacer image, debido a que ExtJS necesita tener la medida de un píxel por un pixel transparente, para dar un ancho fijo a sus componentes. Se debe indicar la locación de esta imagen usando la siguiente línea:

```
Ext.BLANK_IMAGE_URL = 'images/s.gif';
```

Siempre se coloca como primera línea dentro del evento onReady.

La idea de utilizar una spacer image, es porque la interface de usuario de ExtJS es creada usando CSS, pero el CSS necesita elementos HTML fundamentales para crear el estilo que les da la apariencia actual a los componentes de ExtJS. El único elemento HTML que tiene un tamaño exacto y previsible en todos los navegadores en una imagen. Así que esta imagen es usada para definir la forma como los componentes son dibujados. Es así como ExtJS mantiene la compatibilidad con la mayoría de navegadores web.

Ejercicio: Añadir spacer image al ejemplo anterior.

3.6 Probemos los idiomas

ExJS es multi-idioma, el idioma por default es inglés, para cambiarlo solo necesitamos añadir la librería correspondiente al idioma deseado, es así que para español usaremos:

```
<!-- Librería del idioma español-->
<script type="text/javascript" src="ExtJS/src/locale/ext-lang-es.js"></script>
```

Esta declaración debe colocarse antes de las librerías propias.

Ejercicio: Colocar la librería en el index.html y correr el ejemplo con la librería y sin la librería para ver que sucede.

3.7 Algo un poco más complicado

Vamos a crear un ejemplo más complicado, utilizando otros elementos de Ext.Msg, para lo cuál creamos un archivo denominado pregunta.js en el directorio de nuestras librerias, con el siguiente código:

```
Ext.onReady(function()
{
  Ext.BLANK_IMAGE_URL = 'images/s.gif';
  Ext.Msg.show(
  {
    title: 'Javier',
    msg: 'Has visto mi computadora?',
    buttons:
    {
      yes: true,
      no: true,
      cancel: true
    }
  });
});
```

Con este código creamos un cuadro de diálogo con un título definido y cuyo mensaje principal es una pregunta, además le colocamos tres botones como opciones de respuesta.

El argumento pasado a onReady es una función, que puede ser pasada con un nombre definido, o creada en línea como en este caso. Este método de crear una función en línea se denomina como una función anónima, que es usada para llamar una función particular solo una vez.

Si estuviéramos usando una función que pueda ser reultizada, entonces deberíamos definirla de esta forma:

```
function fDondeEsta()
{
  Ext.Msg.show(
  {
    title: 'Javier',
    msg: 'Has visto mi computadora?',
    buttons:
    {
      yes: true,
      no: true,
      cancel: true
    }
  });
}

Ext.onReady(function()
{
  Ext.BLANK_IMAGE_URL = 'images/s.gif';
  fDondeEsta();
});
```

Si estamos pensando realizar una aplicación grande, es buena idea hacer funciones reutilizables.

También se pueden hacer cambios en la configuración de los componentes, colocando estilos propios. Y es posible incluir una función para que se ejecute al hacer clic en los botones del cuadro de diálogo, de la siguiente forma:

1. Se debe crear un archivo de estilos, denominado estilos.css en la carpeta estilos del proyecto.
2. Colocar en el archivo un estilo propio como el siguiente:
```
.persona-icon {
    background: url(../imagenes/ico_persona_flecha.gif) no-repeat;
}
```
3. Colocar el estilo creado en la propiedad *icon* del cuadro de diágolo:
```
icon: 'persona-icon',
```
4. Crear una función anónima asociada a la propiedad *fn* del cuadro de diálogo, esta función hara que aparezca una alerta indicando cual fue el botón que presionó el usuario:
```
fn: function(btn)
    {
      Ext.Msg.alert('Javier hizo clic en', btn);
}
```

3.8 JSON y el objeto de configuración

En nuestros ejemplos, hemos usado algo que se denomina objeto de configuración, que es la forma principal de hacer que ExtJS haga lo que nosotros queremos. Esto permite realizar la configuración de las diferentes opciones que están disponibles para cualquier función que esté siendo usada.

3.8.1 La manera antigua

Antes solíamos llamar a las funciones con un set de argumentos predeterminados. Es decir que debíamos recordar el orden de los argumentos cada vez que la función sea usada.

```
var prueba = new FuncionPrueba('tres','argumentos','usados');
```

Esta manera antigua de crear las funciones puede dar muchos problemas:

- Requiere que recordemos el orden de los argumentos
- No describe lo que cada argumento representa
- Ofrece menos flexibilidad para aumentar argumentos adicionales

3.8.2 La nueva forma de configurar objetos

Usando el objeto de configuración, tenemos un alto nivel de flexibilidad, y se puede decir que nuestras variables están en un plano descriptivo. El orden de los argumentos no importa más, ya que el primero puede ser el último o estar en medio. Con el método del objeto de configuración de pasar argumentos a sus funciones, los argumentos no necesarios puede ser obviados de la declaración.

```
var prueba = new FuncionPrueba(
  {
    primeraPalabra: 'tres',
    segundaPalabra: 'argumentos',
    terceraPalabra: 'usados'
  });
```

Este método también permite una expansión ilimitada de los argumentos de nuestras funciones. Usando pocos o añadiendo más argumentos. Otra gran ventaja de usar el objeto de configuración es que el uso previo de las funciones no se verá afectado por el aumento o disminución de los argumentos en un uso posterior.

```
var prueba = new FuncionPrueba(
  {
    segundaPalabra: 'argumentos'
  });


var prueba = new FuncionPrueba(
  {
    segundaPalabra: 'argumentos',
    cuartaPalabra: 'wow'
  });
```

3.8.3 ¿Qué es el objeto de configuración?

Si se está familiarizado con CSS o JSON, se puede notar que el objeto de configuración luce similar a cualquier de estos dos, sobretodo porque es lo mismo. El objeto de configuración es solo una forma de estructurar datos que puede ser fácilmente interpretada por los lenguajes de programación, en este caso JavaScript.

Por ejemplo, veamos la porción de la configuración del nuestro ejemplo:

```
{
    title: 'Javier',
    msg: 'Has visto mi computadora?',
    buttons:
    {
      yes: true,
      no: true,
      cancel: true
    },
    icon: 'persona-icon',
    fn: function(btn)
    {
      Ext.Msg.alert('Javier hizo clic en', btn);
    }
 }
```

La configuración particular que estamos usando aquí podría parecer completa a primer vista, pero una vez que la conocemos, se convierte en una forma extremadamente fácil de configurar componentes. Casi todos los componenes de ExtJS utilizan el objeto de configuración, así que es algo que se nos tiene que volver familiar. El objeto de configuración se volverá nuestro mejor amigo.

Aquí se detallan algunos puntos importantes que hay que recordar cuando se trabaja con el objeto de configuración:

- Se deben colocar llaves para identificar el conjunto de registros, que simboliza los registros dentro de los corchetes como parte de un objeto > *{registros}*.
- Cada registro se compone de un par nombre/valor, separados por dos puntos, y separados por comas entre ellos > *{nombre0:valor0, nombre1:valuor1}*.
- Los valores del registro puente ser de cualquier tipo de dato, incluyendo bolean, array, function, o cualquier otro objeto > *{nombre0: true, nombre1:{nombre2:valor2}}*.
- Los corchetes definen un arreglo > *{nombre:['uno','dos','tres']}*. Un arreglo puede también contener objetos con registros, valores o cualquier número de otras cosas.

La mayor ventaja de usar JSON para configurar nuestros componentes, es que si nosotros necesitamos más opciones, podemos empezar digitándolas y listo. Contrario a la típica llamada a una función, el orden de nuestras opciones de configuración llega a ser irrelevante, y puede haber pocos o tantos argumentos como sean necesarios.

3.8.4 Como funciona JSON

Todo se basa en la función ***eval***, que es la que JavaScript usa para interpretar una cadena JSON, y convertirla en objetos, arrays y funciones que pueden ser usadas.

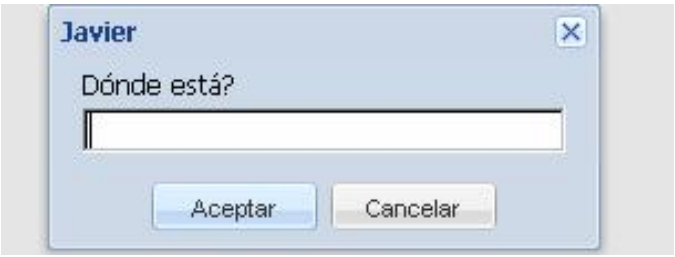
3.9 Modificando el ejemplo anterior

Vamos a abrir nuestro ejemplo pregunta.js y vamos a grabar como pregunta2.js y vamos a añadirle funcionalidad a cada botón de diferente manera:

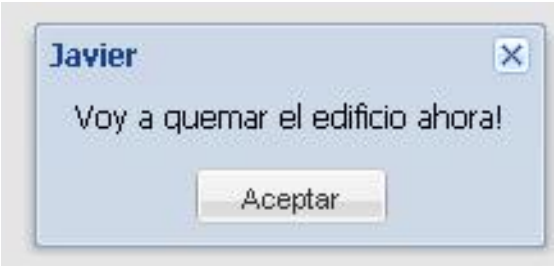
```
fn: function(btn)
    {
      switch (btn)
      {
        case 'yes':
          Ext.Msg.prompt('Javier', 'Dónde está?');
          break;
        case 'no':
          Ext.Msg.alert('Javier', 'Voy a quemar el edificio ahora!');
          break;
        case 'cancel':
          Ext.Msg.wait('Grabando al disco', 'Copia de archivos');
          break;
      }
    }
```

En este ejemplo vemos los diferentes elementos de Ext.Msg,

Cuando se da clic en el botón Si, se despliega un diálogo tipo prompt que permite ingresar un solo valor y es un elemento estándar en casi todas las interfaces de usuario.



Cuando se de clic en No se despliega una alerta, parecida a la alerta estándar del JavaScript pero con esteroides.



Al hacer clic en el botón Cancelar (o presionar la tecla *Escape*) se despliega un mensaje de espera usando una barra de progreso.

La barra de progreso que estamos usando puede ser controlada y desaparecer cuando el proceso se haya culminado. Para este ejemplo se queda ejecutando sin fin.

3.9.1 Encendiendo el fuego

Ahora, vamos a empezar causando algunas reacciones en nuestra página, basándonos en las respuestas de los usuarios a los diálogos. Dentro de nuestra sentencia *switch*, para el botón "Si" colocaremos un tercer argumento en la función *prompt* que se ejecutará cuando el botón "Aceptar" sea presionado. Vamos a validar el texto introducido en el diálogo del prompt, si es igual a "en la oficina" se presentará el texto "No está ahí", caso contrario se imprimirá el texto introducido.

Necesitamos colocar un DIV en el cuerpo de la página

```
<div id='my_id' style='width:100%;height:100%;overflow:hidden;text-align:center;'></div>
```

Y el siguiente código en el case:

```
case 'yes':
    Ext.Msg.prompt('Javier', 'Dónde está?', function(btn, txt)
    {
      if (txt.toLowerCase() == 'en la oficina')
      {
        Ext.get('my_id').dom.innerHTML = 'Mal trabajo';
      }
      else
      {
        Ext.get('my_id').dom.innerHTML = txt;
      }
      Ext.DomHelper.applyStyles('my_id',
      {
        background: 'transparent url(imagenes/computadora.gif) 50% 50% no-repeat'
      });
    });
    break;
```

Para el caso del "No", se desplegará una alerta, que también cambia el estilo del documento cuando el botón es presionado.

```
case 'no':
    Ext.Msg.alert('Javier', 'Voy a quemar el edificio ahora!', function()
    {
      Ext.DomHelper.applyStyles('my_id',
      {
        'background': 'transparent url(imagenes/fire.png) 0 100% repeat-x'
      });
      Ext.DomHelper.applyStyles(Ext.getBody(),
      {
        'background-color': '#FF0000'
      });
      Ext.getBody().highlight('FFCC00',
      {
        endColor: 'FF0000',
        duration: 6
      });
    });
     break;
```

Analizando el ejemplo, podemos ver lo siguiente:

- Podemos enviar como parámetro una función
- Además vemos la utilización de varias sentencias como:
    - Ext.get() > Este método permite el acceso a cualquier elemento del documento HTML colocando como parámetro su ID y permite manipularlo, como hemos visto en el ejemplo.
    - Ext.DomHelper > Esta clase proporciona una capa de abstracción de DOM y apoya de manera transparente a través de la creación de elementos DOM o utilizando fragmentos de HTML. También tiene la habilidad de crear plantillas de fragmentos HTML, a partir de la construcción de su DOM.
    - Ext.getBody() > Retorna el body actual como un elemento de ExtJS.

CAP.4. Panels y TabPanels

4.1 Panel

El Panel es un contenedor que tiene una funcionalidad específica y componentes estructurales que lo hacen el perfecto bloque de construcción para aplicaciones orientadas a interfaces de usuario.

4.1.1 Elaboración de Panels

Todos los componentes de ExtJS tienen su declaración, sus métodos y propiedades, para cuya definición se utiliza el objeto de configuración antes revisado, en el caso del Panel se configura de la siguiente manera:

1. Creamos en la carpeta librerias de nuestro proyecto un archivo que se llame “panel.js”
2. Creamos el Panel directamente asignando el objeto a una variable usando la llamada al método creador Ext.Panel().
3. Al objeto creado le colocamos las siguientes propiedades:
    a. Título
    b. Ancho
    c. Alto
    d. Y lo más importante, donde va a ubicarse, para ello debemos usar la propiedad *applyTo*,ya que este panel podrá ser el contenedor de otros objetos, más no está contenido dentro de otro componente, si así fuera no necesitaríamos esta propiedad.
4. En el index.html llamamos en las librerías propias, solamente al archivo “panel.js” y corremos nuestra aplicación.

Ejercicio: Añadir más propiedades.

4.2 TabPanel

Un TabPanel puede ser usado exactamente como un Panel estándar para propósitos de diseño, con la ventaja adicional de que posee no solo un panel, si no varios manejados por pestañas separadas, estas pestañas pueden colocarse tanto arriba como abajo del panel.

4.2.1 Construyendo nuestro primer TabPanel

Para construir nuestro primer TabPanel, vamos a crearlo dentro del Panel que creamos con anterioridad y de otra forma de las que hemos observado hasta ahora, para crear un componente de ExtJS dentro de otro, lo podemos hacer de dos formas:

1. Asignando el objeto o componente a una variable y colocando esta variable como item o componente de otro, como ya hemos visto.
2. Realizando una creación interna sin asignar a una variable y para hacer referencia a este objeto en cualquier otro lugar usaremos su ID.

Para ello vamos a usar el ejemplo que ya teníamos de la creación del Panel y seguir los siguientes pasos:

1. Lo vamos a renombrar como tab_panel.js.

2. Colocamos el TabPanel como item del panel.
3. Cambiamos la librería llamada en el index.html y corremos la aplicación.

La propiedad de configuración *items*, es un array de objetos que define cada uno de los tabs contenidos en este TabPanel. El título es la única opción que obligadamente necesita ser colocada para visualizar correctamente cada tab.

También necesitamos colocar un tab activo con la propiedad *activeTab*, que es cero para nuestro TabPanel. Este es el índice de los tabs en el panel, de izquierda a derecha empezando a contar desde cero. Esto le indica al TabPanel que el tab de la posición cero debe hacerse activo por defecto, de lo contrario, resultaría en un panel en blanco hasta que el usuario haga clic en una pestaña.

Ejercicio: Colocar las siguientes propiedades de configuración al TabPanel y ver que sucede:

- closable
- disabled
- tabPosition

4.2.2 Manejo de Tabs, métodos que debes conocer

Tenemos algunos métodos especiales para el manejo de tabs, que son:

- **add(*array objeto de configuración*):** Añade dinámicamente un tab al contenedor, como parámetro se coloca el objeto de configuración deseado para el nuevo elemento.
- **setActiveTab(*String/Number item*):** Coloca un tab específico como activo, para ello hay que enviar como parámetro el índice del elemento deseado.
- **getActiveTab():** Retorna el tab activo actual.
- **show():** Muestra físicamente en el contenedor, un elemento recién añadido o que se encuentre oculto.
- **hide():** Oculta solamente la pestaña del tab, no el cuerpo, para ello hay que utilizar el método hideTabStripItem. Es muy útil para cuando se coloca la característica closable: true, debido a que cuando se cierra una pestaña el objeto se autodestruye, si se vuelve a necesitar el tab hay que volver a crearlo y esto es un poco más pesado, por lo que se debe programar el evento beforemove y hacer que solo se oculte, no se autodestruya.
- **hideTabStripItem(*Number/String/Panel item*):** Oculta el cuerpo de la pestaña.

Estos métodos los veremos ejecutados junto con layouts y regions.

CAP.5. Viewports, layouts y regions

5.1. Viewports

El Viewport es un contenedor (se hereda de la clase *Container*) que en sí, tiene una estructura igual que cualquier otro contenedor, pero tiene un lugar especial dentro de Ext JS. El Viewport representa el total de la superficie visible para el usuario, lo que significa que es la ventana del navegador en otras palabras (o para ser más precisos, la porción donde se ejecuta un sitio web, típicamente llamado ventana del navegador).

Cuando se crea un Viewport automáticamente se coloca dentro del *<body>* del documento cargado en el explorador y se acomoda para ocupar toda la ventana del navegador. También está al tanto de los eventos de cambio de tamaño de la ventana del explorador que puedan ocurrir y reacciona apropiadamente. Por esta razón, solo puede existir un Viewport por página.

El Viewport está dotado con una estructura específica de diseño, puede ser BorderLayout, CardLayout o lo que sea, por ejemplo:

```
new Ext.Viewport(
  {
    layout: "fit",
    items: [
    {
      title: "Ventana del navegador",
      bodyStyle: "background-color:#999999;",
      html: "Algún contenido"
    }]
  });
```

5.2 Layouts

Un layout transforma cualquier contenedor en una verdadera aplicación web. Los estilos más ampliamente usados pueden ser encontrados en sistemas operativos como Microsoft Windows, que utiliza borders layouts, regiones de tamaño variable, acordeones, tabs y casi todo lo que se pueda imaginar.

Para mantener apariencia consistente en todos los navegadores, y para proveer características comunes de interfaces de usuario, Ext JS tiene un poderoso sistema de manejo de layouts. Cada sección puede ser manejada y puede ser movida o escondida, y pueden aparecer en un clic, cuando y donde se necesite.

5.2.1 FitLayout

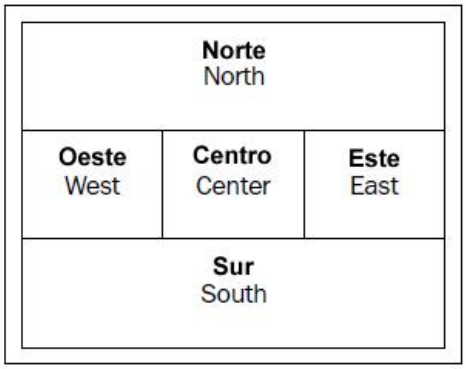
Un FitLayout contiene un solo elemento que automáticamente se expande para llenar todo el contenedor. Por lo general no se crea la instancia de esta clase, en la mayoría de contenedores se coloca el tipo de layout en la propiedad correspondiente.

Curiosamente, FitLayout no tiene sus propias propiedades de configuración, además de las que hereda. Cuando se necesite tener un Panel que llene el contenedor entero, FitLayout debe ser usado. Todo lo que se necesita es colocar el atributo *layout* en el contenedor para rellenar.

5.2.2 BorderLayout

Un BorderLayout es relativamente un layout muy simple que también es de uso muy común, talvez el más popular. Ese es un layout que tiene cinco regiones:

- una a lo largo de toda la parte superior de la página,
- otra en el lado izquierdo,
- otra en el lado derecho,
- otra en la parte de abajo
- y otra entre todas las demás en el medio.



The BorderLayout soporta barras de separación entre las regiones, permitiendo al usuario cambiarlas de tamaño. También soporta secciones que se pueden tanto expandir como colapsar. Tal como FitLayout, el BorderLayout en si mismo no tiene sus propias opciones de configuración.

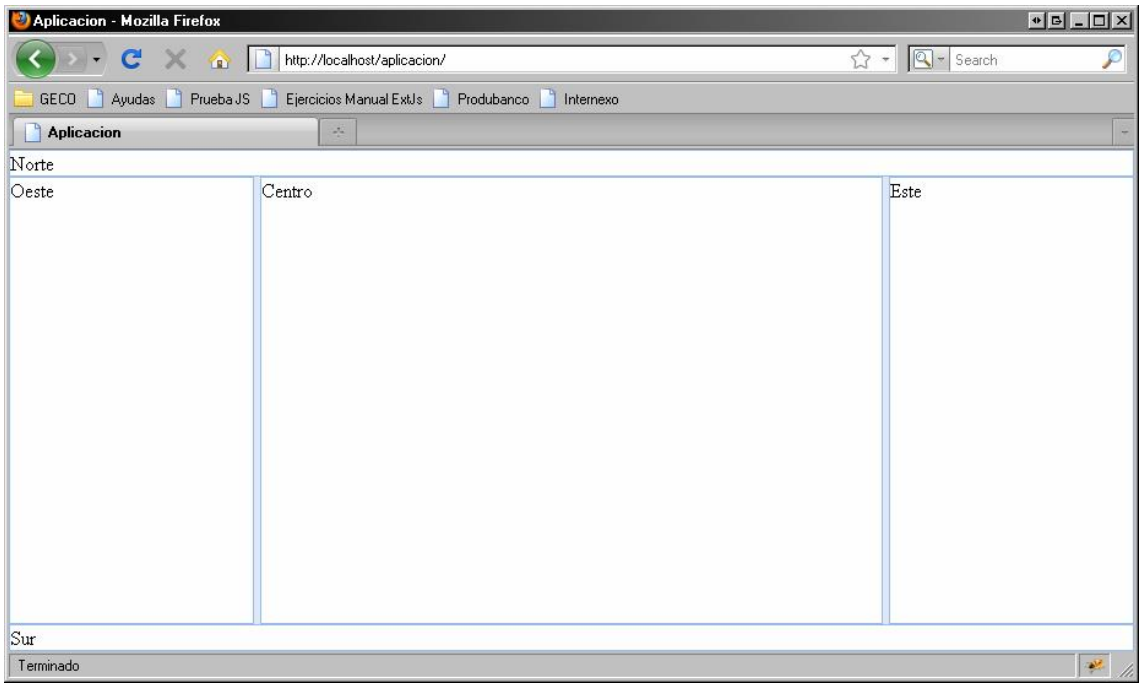
Si bien hay cinco regiones disponibles en el BorderLayout, no es necesario que se usen todas. Sin embargo, no se puede adicionar regiones después de que se ha hecho el BorderLayout, por eso hay que asegurarse de cuantas regiones se necesitan para configurarlas cuando se crea el BorderLayout.

No se puede crear una instancia de *Ext.layout.BorderLayout*. En su lugar se debe configurar el atributo *layout* de algun contenedor. Por ejemplo, creamos un Viewport, para desplegar los usos de BorderLayout para organizar este contenedor:

```
var viewport = new Ext.Viewport(
  {
    layout: 'border',
    renderTo: Ext.getBody(),
    items: [
    {
      region: 'north',
      xtype: 'panel',
      html: 'Norte'
    },
    {
      region: 'west',
      xtype: 'panel',
      split: true,
      width: 200,
      html: 'Oeste'
    },
    {
      region: 'center',
      xtype: 'panel',
      html: 'Centro'
    },
    {
      region: 'east',
      xtype: 'panel',
      split: true,
      width: 200,
      html: 'Este'
    },
    {
      region: 'south',
```

```
      xtype: 'panel',
      html: 'Sur'
    }]
});
```

Si todo salió bien, veremos algo como lo siguiente:



Ahora tenemos un layout con cinco regiones definidas. Estas regiones pueden contener otros componentes, sin problemas. Como alternativa cada región en si misma puede ser dividida en más regiones anidadas, por ejemplo la sección del centro puede ser dividida horizontalmente para tener su propia región sur.

Ejercicio: Hacer que las regiones pueda cambiarse de tamaño o se puedan contraer.

5.2.3 Accordion

El Accordion es en cierto modo su propio componente. En otras librerías realmente lo es, pero en Ext JS es un tipo de layout (es literalmente una extensión de FitLayout). En pocas palabras, un Accordion es una sola capa en la que se tienen diferentes paneles que pueden ser seleccionados por el usuario. Estos paneles están apilados verticalmente (aunque hay algunas implementaciones que permiten hacerlo horizontalmente también) y usualmente incluye un bonito efecto animado cuando se selecciona uno de ellos. Por ejemplo:

Un ejemplo que crea un Accordion puede ser:

```
var panel = new Ext.Panel(
  {
    title: 'Mi primer acordeón',
    layout: 'accordion',
    height: 400,
    collapsible: true,
    layoutConfig:
    {
      animate: true
    },
    items: [
    {
      title: 'Panel 1',
      html: 'Yo soy el Panel #1'
    },
    {
      title: 'Pane 2',
      html: 'Yo soy el Panel #2'
    },
    {
      title: 'Pane 3',
      html: 'Yo soy el Panel #3'
    }]
  });
```

Aquí se crea una instancia de Ext.Panel, con un layout tipo Accordion. Se especifica el alto y se establece la propiedad *collapsible: true*, para que se puedan colapsar y se configura la propiedad *animate: true*, común para todos los paneles, que permite establecer un bonito efecto el momento de seleccionar algún panel.

5.2.4 CardLayout

El CardLayout es un tipo de FitLayout con esteroides, ya que permite tener múltiples paneles adaptados a los contenedores, pero solamente permite mostrar uno a la vez. Interfaces como los wizards son típicamente implementadas con CardLayout, así como las interfaces con pestañas.

El método más importante expuesto por el CardLayout es setActiveItem (). Este permite mostrar un nuevo panel en el CardLayout ya sea por su código por el valor del índice. Este está completamente bajo el control del programador, el CardLayout no puede intercambiarse entre

paneles por si solo (o en respuesta a algún evento de usuario, a menos que se escriba el código para hacerlo), así que no hay algo por defecto para que se intercambien los paneles como es el caso del Accordion.

El CardLayout también soporta la opción de configuración deferredRender, que, cuando es seteada a true, le dice al contenedor que solamente coloque el panel que actualmente se muestra. Esta es una buena opción de configuración para que la carga sea eficiente. Ejemplo:

```
var panel = new Ext.Panel(
  {
    title: 'Mi Primer CardLayout',
    layout: 'card',
    height: 400,
    id: 'myCardLayout',
    activeItem: 0,
    items: [
    {
      title: 'Panel 1',
      html: "Hola, soy el Panel #1<br><br>" +
      "<input type=\"button\" value=\"Clic para cambiar al panel #2\" " +
      "onClick=\"Ext.getCmp('myCardLayout')" +
      ".getLayout().setActiveItem(1);\">"
    },
    {
     title: 'Panel 2',
      html: "Hola, soy el Panel #2<br><br>" +
      "<input type=\"button\" value=\"Clic para cambiar al panel #1\" " +
      "onClick=\"Ext.getCmp('myCardLayout')" +
      ".getLayout().setActiveItem(0);\">"
    }]
  });
```

En este ejemplo tenemos configurado un Panel tipo CardLayout que tiene dos paneles internos, el panel activo es el panel cero, y en la configuración html de cada uno, se crea un botón tipo HTML simple, el cual al presionarlo cambia al otro panel configurado respectivamente.

Nótese que para configurar la propiedad html se usa HTML simple y se maneja el evento onClick del botón dentro del cual se coloca código propio de Ext JS.

5.2.5 TableLayout

Un TableLayout permite crear capas basadas en tablas con facilidad. En algunos casos es conceptualmente similar al BorderLayout excepto que se tiene el control de cuantas regiones podamos querer.

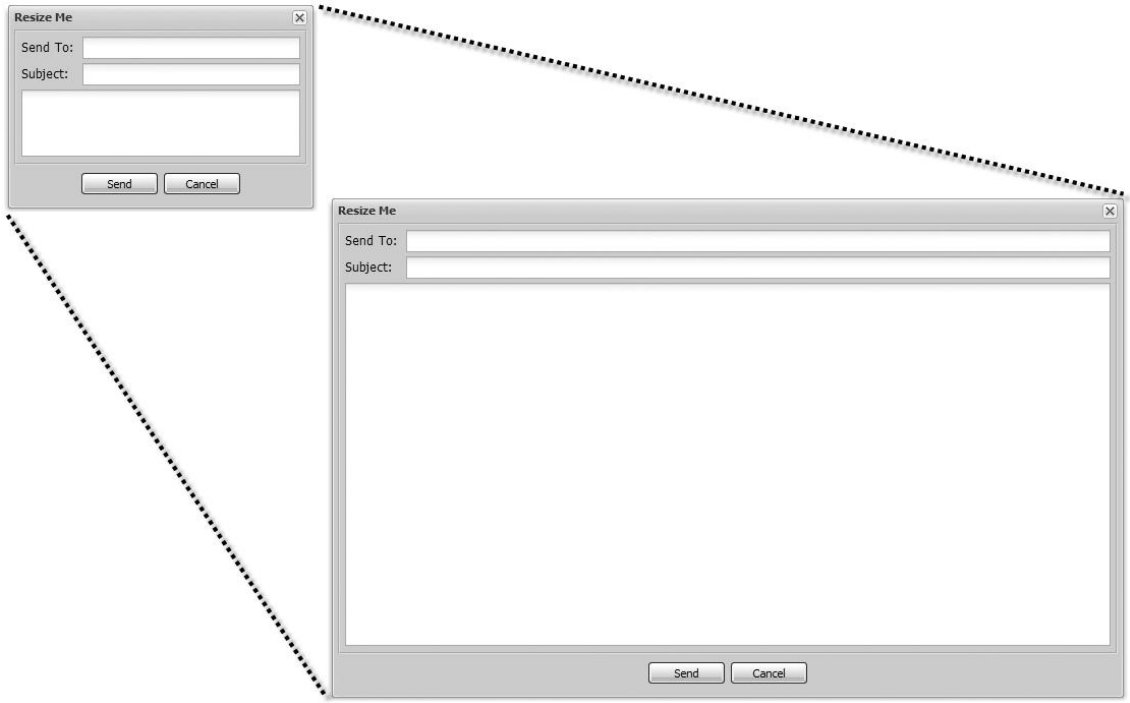
Una gran diferencia entre crear un layout usando TableLayout y usando tablas con HTML plano es que con TableLayout es que ya no nos preocupamos de las tablas y filas de forma explícita. No hay que preocuparse de las celdas con filas ni nada por el estilo. Todo lo que hay que hacer es especificar el número de columnas que el TableLayout debe tener y entonces añadir objetos en cada uno, de derecha a izquierda, de arriba hacia abajo. El TableLayout se dará cuenta de la posición de cada panel basándose en la cuenta de filas, además se podrá colocar cualquier tamaño de fila y columna que se necesite. Si se está acostumbrado a la creación de tablas HTML, utilizando TableLayout puede ser un poco difícil de entender para el cerebro, pero una vez que lo hace, rápidamente, uno se da cuenta de la flexibilidad que ofrece. Ejemplo:

```
var panel = new Ext.Panel(
  {
    title: 'Mi primer TableLayout',
    layout: 'table',
    height: 400,
    layoutConfig:
    {
      columns: 2
    },
    items: [
```

```
    {
      html: 'Columna 1, Celda 1',
      width: 200,
      height: 200
    },
    {
      html: 'Columna 2',
      rowspan: 2,
      width: 200,
      height: 400
    },
    {
      html: 'Columna 1, Celda 2',
      height: 200
    }]
  });
```

5.2.6 AnchorLayout

Un AnchorLayout es un layout que permite que los elementos colocados en un contenedor se acoplen unos con otros. En otras palabras, si el contenedor cambia de tamaño, ya sea porque el contenedor mismo cambia de tamaño o indirectamente del resultado del cambio de su contenedor principal, entonces todos los elementos dentro de él se acoplan al nuevo tamaño y lo más importante se redimensionan, de acuerdo a las reglas que se configuren.



En esta figura se puede ver que al cambiar de tamaño a la ventana, los componentes adentro también cambian de tamaño. Ejemplo:

```
var ventana = new Ext.Window(
  {
    resizable: true,
    layout: "anchor",
    title: "Mi Primer AnchorLayout",
    width: 200,
    height: 200,
    items: [
    {
      xtype: "textfield",
      anchor: "100%",
      value: "textfield1"
    },
```

```
    {
      xtype: "textfield",
      anchor: "100%",
      value: "textfield2"
    },
    {
      xtype: "textarea",
      anchor: "100% 60%",
      value: "textarea"
    }]
  }).show();
```

En este caso tenemos dos campos de texto y un área de texto. En la ventana se especifica el atributo layout como anchor y en cada elemento también. El valor de este atributo es siempre en la forma “xx yy”; donde xx es el valor de anclaje horizontal y yy es el vertical.

Tres tipos de valores son soportados aquí, El primero es un porcentaje. Así que en el código del área de texto el atributo anchor le dice que debe ser expandido para rellenar la ventana horizontalmente y que puede tomar un 60% del área disponible en la ventana. Se puede configurar también un solo valor que especifica el alto y el ancho automáticamente.

Se puede también especificar un valor de desplazamiento para el atributo anchor. Este puede ser positivo o negativo. El primer valor es un desplazamiento desde la derecha del contenedor, y el segundo desde abajo. Así, si el atributo anchor para el área de texto fuera -25 -75, indica que se debe dibujar el item al ancho completo de la ventana menos 25 pixels y el alto completo de la ventana menos 75 pixels. Al igual que con porcentajes, en su lugar sólo puede especificar un valor único, y que se tomará como el derecho de compensación, con el fondo compensado por defecto a 0.

Se puede también especificar un valor anchor para derecha, o r, otro para abajo o b. Para que esto haga algo, sin embargo, el contenedor debe tener un tamaño arreglado o debe ser configurada la propiedad anchorSize.

También se puede mezclar ambos tipos de valores, por ejemplo un valor de -10 80% significa que el elemento debe ser dibujado al ancho completo del contenedor menos 50 pixels desde la derecha y al 80 por ciento del alto del contenedor.

CAP.6. Ventanas y diálogos

6.1 El ayer y el hoy con ventanas

En tiempos pasados de la web, los usuarios de los sistemas tradicionales pasarían su tiempo ingresando datos en listas y formas. Escogiendo ítems de una lista de órdenes de compra, luego navegando hacia los detalles una y otra vez. El problema es que estamos hablando de cientos de entradas en una lista y varios detalles en las formas. Lo más probable era que, en el ejemplo anterior de órdenes de compra, se necesita más sub-formas para mostrar toda la información que está disponible y cada vez que el usuario se mueve a otra pantalla se tenga que refrescar la página completa y traer todos los datos de nuevo de la base de datos.

Hoy con los objetos y herramientas que Ext JS nos provee, podemos realizar un mejor trabajo, tanto en presentación y obtención de datos como en velocidad de acceso a las pantallas.

Otra parte importante del rompecabezas que nos permite hacer esto son las ventanas y los diálogos, ya que nos permiten presentar cualquier clase de información sin forzar a los usuarios a navegar a otras pantallas.

6.2 Diálogos

Los diálogos son una poderosa herramienta para mostrar alertas, mensajes y errores al usuario, haciendo un buen uso de ellos logramos tener una buena comunicación entre el sistema y el usuario creando una interfaz amigable y cómoda.

Todos los diálogos de Ext JS se toman de la clase *Ext.MessageBox* o con el alias *Ext.Msg*.

6.2.1 Alert

Ext JS nos provee con un excelente reemplazo a las alertas simples de JavaScript. Con la siguiente línea de código veremos como funciona:

```
Ext.Msg.alert('Hey!', 'Algo pasa!');
```

La primera cosa que notamos es que *Msg.alert* tiene dos parámetros, mientras que la alerta estándar tiene solo uno. El primero nos permite especificar el título de la alerta y el segundo especifica el cuerpo. Lo que se despliega al ejecutar el código anterior es:

Como se puede ver, su funcionamiento es el mismo que la alerta estándar pero tiene una mejor apariencia y es más versátil. Podemos también transmitir más información con la barra del título. Mostrar *Msg.alert* no detiene temporalmente la ejecución del script como lo hace la alerta normal; hay que ser concientes de esto cuando se usa la versión de Ext JS.

También hay que tomar en cuenta que se puede usar solo un *Ext.MessageBox* al mismo tiempo. Si se trata de enviar dos al mismo tiempo, el primero será reemplazado por el segundo.

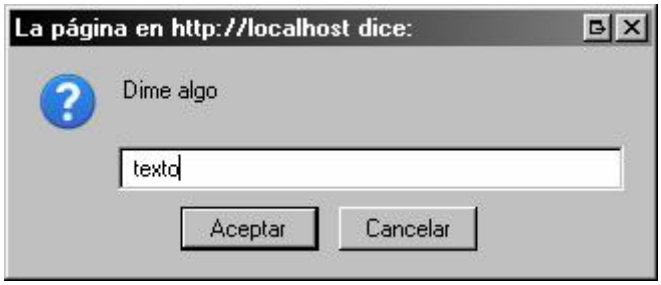
6.2.2 Prompt

Otro componente tipo diálogo es *Ext.Msg.prompt*. Este nos permite capturar una simple línea de texto de la misma forma que el prompt estándar de JavaScript. Sin embargo, en lugar de limitarse a devolver un valor, nos da algunas opciones más.

Ejemplo:

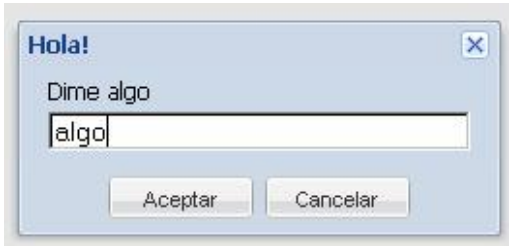
Con JavaScript estándar:

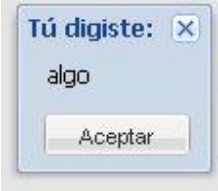
```
<script type="text/javascript">
   var data = prompt('Dime algo');
   alert(data);
</script>
```



Con Ext JS:

```
Ext.Msg.prompt('Hola!', 'Dime algo', function(btn, text)
  {
    if (btn == 'ok')
    {
      var data = text;
      Ext.Msg.alert('Tú digiste:', data);
    }
    else
      Ext.Msg.alert('Ohh', 'No has querido decir nada');
  }, this, false, '');
```





Una vez más, *Msg.prompt* permite pasar un título como el primer argumento, y el cuerpo del texto es el segundo. El tercer argumento es una función de devolución de llamada o callback function en inglés, que puede ser llamada cuando cualquiera de los dos botones, Aceptar o Cancelar sea presionado. La función tiene dos argumentos, el botón que fue presionado y el texto que fue ingresado por el usuario.

Los otros tres parámetros de la función, son el objeto, un indicador de multilínea y un valor inicial, respectivamente. El argumento de multilínea permite tener un área de escritura más amplia para el prompt.

6.2.3 Confirmation

El diálogo de confirmación permite al usuario escoger entre una acción de confirmación o rechazo a alguna acción. El código es el siguiente:

```
Ext.Msg.confirm('Hola!', 'Está de acuerdo?', function(btn, text)
  {
    if (btn == 'yes')
    {
      Ext.Msg.alert('Bien', 'Me alegro mucho');
    }
    else
    {
      Ext.Msg.alert('Está bien', 'No hay problema');
    }
  });
```

De nuevo usamos, un título, un texto en el cuerpo y una callback function como en el prompt. Una interesante diferencia con la confirmación estándar de JavaScript es que da las opciones de Aceptar y Cancelar y no solo Si y No.

6.2.4 Progress

Los anteriores tipos de diálogos eran un reemplazo a lo que ya existía en el JavaScript estándar, ahora vamos a ver un diálogo más, desarrollado por Ext JS, que es el diálogo de progreso. *Ext.Msg.progress*, este no está diseñado para ser usado independientemente como los otros diálogos, y no necesita acción del usuario. De hecho, se puede disparar de la siguiente forma:

```
Ext.Msg.progress('Hola!', 'Estamos esperando...', 'progreso');
```

Con esta declaración estaremos esperando hasta que nos cansemos, porque es un diálogo que nunca progresa. Los dos primeros argumentos son el título y el texto del cuerpo del mensaje, igual que en los anteriores tipos de diálogos, mientras que el tercero es el texto que aparecerá en la barra de progreso.

Así que para no esperar eternamente, debemos actualizar el movimiento, para eso nos ayuda el método *Ext.Msg.updateProgress*, creado solo para este propósito. A continuación un ejemplo de su uso:

```
var count = 0;
  var interval = window.setInterval(function()
  {
```

```
    count = count + 0.04;
    Ext.Msg.updateProgress(count);
    if (count >= 1)
    {
      window.clearInterval(interval);
      Ext.Msg.hide();
    }
  }, 100);
```

Este es un ejemplo muy artificial, en el que llamamos al método *upgrateProgress* cada 100 milisegundos a través de un temporizador, e incrementa el progreso usando la variable *count* cada vez. El primer argumento de este método es un valor entre cero y uno, con cero representamos el inicio y con uno representamos el fin, el segundo permite actualizar el texto de la barra de progreso y el tercero permite cambiar el texto del cuerpo del mensaje. Actualizar el texto puede ser útil si se desea proveer de información adicional al usuario, o para mostrar la representación del porcentaje completado del proceso actual.

Regresando al ejemplo, nótese que se hace referencia también a *Ext.Msg.hide*, con el fin de limpiar la barra de progreso de la pantalla, ya que el método *updateProgress* no maneja esto, incluso si se establece el progreso actual a un valor de más de uno.

6.2.5 Show

Los cuatro métodos anteriores para crear diálogos, son en esencia accesos directos hacia un quinto método: *Ext.Msg.show*. Este método toma un objeto de configuración como su único argumento y las opciones de configuración dentro de él permiten la creación de un messagebox que soporta todas las características disponibles a través de los métodos anteriores. La forma más simple de este método es:

```
Ext.Msg.show(
  {
    msg: 'IMPRESIONANTE.'
  });
```

Esta es una réplica más cercana a la alerta estándar de JavaScript, pero no es funcional, algo mejor sería:

```
Ext.Msg.show(
  {
    title: 'Hola!',
    msg: 'Iconos y Botones! IMPRESIONANTE.',
    icon: Ext.MessageBox.INFO,
    buttons: Ext.MessageBox.OK
  });
```

Ahora tenemos otra vez un título, un texto en el cuerpo, pero ahora hay un icono y un solo botón.

La forma de configurar iconos y botones es interesante, se pasa una de las constantes que Ext JS provee y se puede tener un botón pre-configurado o también se puede una utilizar una clase CSS que muestre un icono. La lista de constantes para iconos es:

- Ext.Msg.ERROR
- Ext.Msg.INFO

- Ext.Msg.QUESTION
- Ext.Msg.WARNING

Y las constantes para botones son:

- Ext.Msg.CANCEL
- Ext.Msg.OK
- Ext.Msg.OKCANCEL
- Ext.Msg.YESNO
- Ext.Msg.YESNOCANCEL

Esta variedad de opciones listas provee gran flexibilidad cuando deseamos colocar una apariencia gráfica a nuestros messageboxes, pero podemos hacer más cosas. Las mencionadas constantes de iconos son simplemente cadenas que representan nombres de clases elaboradas con CSS. Por ejemplo, *Ext.Msg.QUESTION* tiene por detrás la configuración *ext-mb-question*, que no es más que una clase configurada de CSS. Esto se encuentra en las hojas de estilo de Ext JS. La conclusión lógica es que se puede también configurar nuestras propias clases CSS para colocar en lugar de estas constantes, lo que permite gran personalización de las áreas para iconos.

Las constantes de botones son un poco menos flexibles, y contienes objetos literales especificando como deben ser desplegados. Por ejemplo, *Ext.Msg.YESNOCANCEL* contiene lo siguiente (representado en JavaScript Object Notation para fácil lectura):

```
{cancel:true, yes: true, no:true}
```

Esto especifica que los botones, yes, cancel y no deben ser incluidos. Se puede usar esto para apagar un botón u otro, pero no se puede cambiar el orden en el que aparecen, debido a que es un estándar.

Sin embargo, podemos ajustar los diálogos de otras maneras. Es posible colocar dimensiones a un cuadro de diálogo, alto y ancho. Este puede ser útil en situaciones donde se tenga que desplegar un mensaje muy grande y se deba evitar que se extienda a lo largo de la pantalla.

El cuadro de diálogo *show*, y su objeto de configuración permiten la opción *cls* para especificar una clase CSS para aplicar al contenedor del diálogo. Un desarrollador podría usar esto para remarcar cualquier objeto del contenedor usando reglas CSS.

6.2.6 Comportamiento general de Show

Hasta el momento, las opciones de configuración para *Ext.Msg.show* han sido solo de apariencia, pero hay algunas opciones que pueden ajustar el comportamiento. Si se usa la propiedad *progress*, entonces se puede replicar el diálogo estándar de progreso de Ext JS:

```
Ext.Msg.show({progress:true});
```

Mediante el uso de este conjunto de opciones con otras como título y cuerpo de mensaje, se puede crear cuadros de diálogo bastante personalizados.

De forma similar, el prompt y las opciones de multilínea permiten la creación de cuadros de diálogo de entrada:

Así que se puede crear un cuadro de diálogo que acepte múltiples líneas de ingreso. Pero por omisión el valor multilínea es falso, se puede limitar el cuadro de diálogo a una sola línea, etc. Se puede hacer cuadros de diálogo de entrada muy personalizados.

Otra opción que cambia el comportamiento por defecto de un cuadro de diálogo es *modal*. Esta opción permite especificar si el usuario puede interactuar con los elementos que están detrás

de la ventana emergente. Cuando se establece en *true*, se coloca un recubrimiento semitransparente para evitar la interacción.

Como hemos visto antes, los cuadros de diálogo de Ext JS, no bloquean la ejecución del script, como sucede en JavaScript. Esto significa que se pueden usar callback functions para disparar código después de que el diálogo es cerrado. Podemos hacer esto usando la opción de configuración *fn* de *show*, que es llamada con dos argumentos, el código del botón que ha sido presionado y el texto ingresado dentro del campo en el diálogo (donde el diálogo incluye un campo de entrada). Obviamente, para una alerta simple el parámetro del texto vendrá en blanco, pero esta función proporciona de forma consistente toda la gama de cuadros de diálogo que pueden ser creados utilizando *Ext.Msg.show*.

6.3 Ventanas

Cualquier usuario de computadoras esta familiarizado con el concepto de ventanas; un panel informativo que aparece en la pantalla para proporcionar más datos sobre las acciones del usuario actual. Ext JS aplica este concepto mediante el uso de la clase *Ext.Window*, un poderoso componente que soporta varios avanzados escenarios.

6.3.1 Empezando

Se puede abrir una ventana usando una pequeña cantidad de código:

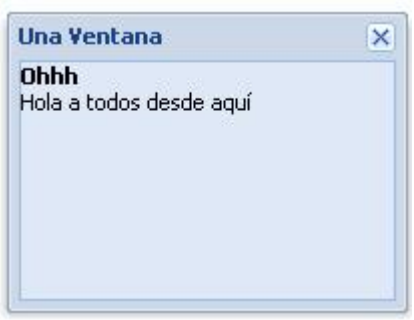
```
var w = new Ext.Window(
  {
    height: 100,
    width: 200
  });
w.show();
```

Corriendo este código se obtiene una ventana vacía que en si misma es completamente inútil, pero esto muestra algunas de las características predeterminadas de *Ext.Window*. Así, sin ninguna configuración, la ventana se puede arrastrar y se puede ajustar en tamaño, además se tiene un útil icono para cerrarla en la esquina superior derecha. Todavía no es una demostración muy impresionante, sin embargo, porque nuestra ventana en realidad no muestra nada.

La forma más fácil de rellenar una ventana es con el viejo y plano HTML. Por ejemplo:

```
var w = new Ext.Window(
  {
    height: 150,
    width: 200,
    title: 'Una Ventana',
    html: '<h1>Ohhh</h1><p>Hola a todos desde aquí</p>'
  });
  w.show();
```

Se han añadido dos nuevas opciones de configuración aquí. La primera es un título que permite colocar un texto en la barra superior de la ventana y la segunda permite aceptar una cadena HTML que despliega un texto en la ventana.



El uso de este enfoque es inmediatamente evidente, se puede volver a lo básico e inyectar cualquier contenido HTML que se requiera directamente en el área del contenedor. Esto nos permite modificar nuestra ventana justo cerca del nivel marcado y proporcionar cientos de CSS para enganchar el estilo. Aún así, esto no es lo que se espera llegar a tener con Ext JS. Hay muchas otras opciones que permitirán llegar a lo deseado.

6.3.2 Un panel con potencia

Hay que recordar que *Window* es una subclase de *Panel*, y *Panel* tiene toda clase de interesantes trucos bajo la manga. La opción de configuración *ítems*, acepta un vector de objetos de configuración u otras instancias de componentes:

```
var w = new Ext.Window(
  {
    items: [
    {
      xtype: 'textfield',
      fieldLabel: 'Nombre'
    }, new Ext.form.TextField(
    {
      fieldLabel: 'Apellido'
    })]
  });
  w.show();
```

En el ejemplo de arriba, se han añadido dos *textfields*, el primero usando inicialización “lazy” con *xtype* y el segundo usando una inicialización estándar. Estos dos ítems serán añadidos al panel interno de la ventana, pero la manera en la que son desplegados puede ser controlada con la propiedad *layout* de nuestra ventana.

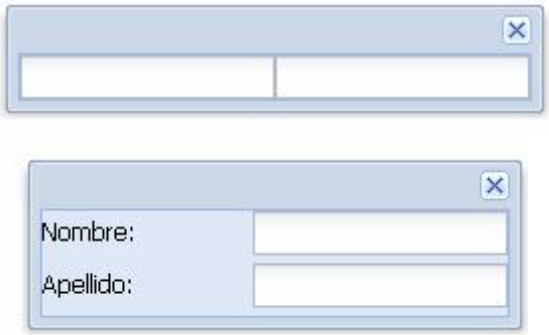
6.3.3 Layout

Ext JS define un número de modelos de *layout* dentro del paquete *Ext.layout* y cada uno de estos puede ser con un panel para permitir que los componentes dentro de él sean dispuestos de una manera específica. En el ejemplo anterior, se añadieron dos cajas de texto a la ventana, pero podemos mejorar la apariencia de la ventana simplemente usando el *layout* apropiado. En este caso, necesitamos *Ext.layout.FormLayout*, que proporciona soporte para etiquetas y el espacio general y el posicionamiento esperado para una forma con campos editables:

```
var w = new Ext.Window(
  {
    layout: 'form',
    items: [
    {
      xtype: 'textfield',
      fieldLabel: 'Nombre'
    }, new Ext.form.TextField(
    {
      fieldLabel: 'Apellido'
    })]
```

```
  });
  w.show();
```

Usando la opción de configuración *layout*, para especificar que se desea tener un panel tipo *form*, se puede apreciar inmediatamente la diferencia:



Esta no es una característica de *Ext.Window*, ya que proviene de la super clase Panel. Pero el hecho de que una ventana soporte esta característica es extremadamente importante para un desarrollador de aplicaciones, especialmente si tenemos en cuenta cuánto tiempo tomaría crear una forma rica con la técnica de inyección de código HTML. Las otras *layouts* dentro del paquete *Ext.Layout*, proporcionan muchos más enfoques de diseño y expansión de escenarios que una ventana puede soportar.

Además para las varias maneras de rellenar un área de contenido de una ventana, también se tiene una gran flexibilidad cuando se trata de la apariencia y el comportamiento de un cuadro de diálogo. Hay muchas opciones de configuración proporcionados por la jerarquía de la super clase *Ext.Window*, que empiezan con el *Ext.Panel*, al mismo tiempo que tiene una gran cantidad de opciones de configuración propias.

6.3.4 Limpiando ventanas

En nuestro primer ejemplo de ventanas, se demostró que se tiene un gran número de características para reajustar, arrastrar libremente y además se tiene un botón para cerrar. Ya dentro de una aplicación, pueden haber ventanas con estrictos diseños de configuración, y no necesitan ser reajustables, por lo que se puede evitar este comportamiento seteando el valor *false* en la propiedad *resizable*. A menudo arrastrar, también es solo una cuestión de preferencia, y en muchos casos dejar activada esta propiedad puede causar daño y sería mejor deshabilitarla. Lo que quiere decir que hay veces en las que no es necesariamente funcional, para mostrar como, tenemos el siguiente ejemplo:

```
var w = new Ext.Window(
  {
    height: 50,
    width: 100,
    closable: false,
    draggable: false,
    resizable: false
  });
w.show();
```

Cuando se usa una ventana tipo forma, a menudo es preferible disponer de botones de texto para explicar las diferentes acciones, por ejemplo, se puede grabar un registro o cancelar algún cambio que haya sido realizado y en algunos casos, el botón cerrar puede ser deshabilitado, colocando la opción *closable* a *false*. Hay una segunda opción que da un poco más de control sobre este comportamiento: *closeAction* puede ser configurada para cuando deseamos ocultar y no cerrar nuestra ventana, con *hide* simplemente se hace que desaparezca la ventana, mientras que con *close*, se remueve la ventana del DOM. Esta es una importante diferencia si

se desea reutilizar la ventana después, ya que esconder la ventana para mostrarla cuando se necesite es más eficiente que recrearla cada vez.

6.3.5 Los extras

Con las opciones de configuración bajo control, se pueden revisar las maneras en las que se puede modificar y aumentar las características de la *Ext.Window*. Ya se ha demostrado el uso de las opciones *height* y *width*, para configurar las dimensiones de la ventana y recortar cualquier contenido que exceda estas dimensiones.

Se tienen también otras opciones como *autoheight* y *autoWidth*, que reciben valores buleanos, que permiten rellenar la ventana con componentes sin tener que preocuparse de asegurar que los valores de ancho y alto sean siempre correctos. Esto es realmente útil durante el desarrollo, cuando es poco probable que se tengan en cuenta las dimensiones exactas de lo que se está creando, solo se debe colocar *true* en las propiedades *autoheight* y *autoWidth.* Aún mejor, estas opciones se pueden usar separadamente, lo que permite colocar un ancho fijo pero se puede colocar un largo ajustable y viceversa. Esto es útil si se está colocando contenido dinámico en la ventana, pero en este caso, hay que estar seguros de las dimensiones de la ventana no excederán los lados de la pantalla.

6.3.6 Funcionando dentro de un escritorio

El ejemplo más generalizado de un sistema de ventanas es el que nos muestra un escritorio de computadora, con varias ventanas que representan aplicaciones o elementos del sistema de archivos. En dichos sistemas, los usuarios pueden esconder ventanas para usarlas después, o puede minimizarlas; ellos también son capaces de expandir las ventanas hasta llenar la pantalla o maximizarlas. Estos son términos familiares y por supuesto son soportados por *Ext.Window* a través de las opciones de configuración *maximizable* y *minimizable*.

Estas características están deshabilitadas por defecto, pero pueden ser de utilidad para trabajar en ambientes tipo escritorio. Cuando se configuran con true, van a aparecer nuevos iconos en la parte superior derecha de la ventana de forma similar a las ventanas del sistema operativo Ms. Windows. La propiedad *maximizable* permite que la ventana se expanda y llene todo el espacio disponible en la ventana del explorador, como se espera, pero la propiedad *minimizable*, es un poco más difícil. Ext JS no sabe donde se va a esconder la ventana minimizada, que en el caso del sistema operativo Ms. Windows sería a la barra de tareas, pero para otros sistemas operativos podría ser en otro lugar. Por lo que se debe programar la funcionalidad de minimizar a mano, Ext JS provee solo un icono para el minimizado y el evento minimizar que debe ser manejado de manera apropiada para la aplicación que se esté desarrollando, para minimizar una ventana de Ext JS, hay que utilizar CSS y programación adicional.

6.3.7 Otras opciones

La clase Ext.Window, tiene otros medios para cambiar el estado real de las ventanas, y está integrado todo dentro del framework. La propiedad buleana *collapsible*, añade otro botón a la parte superior derecha de la ventana y permite al usuario encogerla para que se muestre solo la barra superior. Un segundo clic expande la ventana, regresándola a su estado original.

Se puede usar también la configuración *expandOnShow* para especificar que una ventana escondida siempre debe aparecer expandida a toda su dimensión cuando se muestre de

nuevo. Esto es útil para las ventanas que han sido escondidas previamente y necesite ser traídas de vuelta.

Además de la barra de título estándar y el área de contenido del cuerpo, se puede añadir más áreas de contenido a una ventana. Algunas de estas áreas pueden ser completamente personalizadas, y algunas son un poco más restrictivas, pero juntas estas propiedades son otro método de crear ventanas funcionales.

Dependiendo de los requerimientos, se puede escoger usar una o más de estas áreas de contenido para proporcionar herramientas que permitan a los usuarios manipular y consumir la información dentro de las ventanas. Un típico ejemplo podría ser crear una ventana con un layout tipo form, que incluye los botones de Grabar y Cancelar en el pie de la misma. Esto refleja el estilo típico de una forma de entrada de datos, y puede ser posicionada automáticamente por medio de Ext JS con una pequeña configuración.

6.3.8 Manipulación

Cuando nuestras ventanas están en la pantalla, tenemos un rango de métodos que pueden ser usados para cambiar su posición, apariencia y comportamiento. De hecho, ya hemos usado uno de estos métodos en nuestros ejemplos –show- que es usado para desplegar la ventana en primer lugar. Aunque hemos usado *show* en la mayoría de ejemplos, este método puede tener tres argumentos, todos son opcionales.

```
w.show('animTarget', function()
  {
    alert('Ahora mostrando');
  }, this);
```

Primeramente, podemos especificar une elemento, o el ID de un elemento, para formar el punto de partida desde el que la ventana deberá animarse cuando se abra. Este efecto cosmético puede también ser especificado usando la opción de configuración *animateTarget*. El segundo argumento es una callback function, que se dispara cuando la presentación de la ventana se ha completado, y el tercer argumento es simplemente el ámbito definido para la función que se ejecutará. Resultó que el método *show* no era tan simple después de todo.

El obvio compañero de *show* es *hide*. En efecto, este toma los mismos argumentos, y hará que la ventana desaparezca de la pantalla para su posterior uso. Si se está seguro de que no se necesitará la ventana nuevamente entonces es probable usar el método *close*, que remueve a la ventana del DOM y la destruye.

La funcionalidad de los métodos *close* y *hide*, se manejan para los iconos de la ventana. Hay un poco más de métodos que permiten tener control sobre los ítems, que ya hemos visto como son *minimize* y *maximize*. Esta funcionalidad básica se aumenta con el método *restore*, que es usado después de maximizar y retorna la ventana con sus dimensiones originales, y *toggleMaximize*, que es simplemente un acceso directo entre *maximize* y *restore*. Y en términos de configuración, para que la ventana retorne a su configuración inicial, usamos el método *center*, que coloca la ventana en el medio de la ventana del explorador.

También se puede manipular la posición de nuestras ventanas, *alignTo* permite a un desarrollador por medio de código, mover una ventana próxima a otro elemento especificado.

Este método tiene tres parámetros:

- El elemento al que se va a alinear la ventana.
- La posición de alineamiento
- Una posición de desplazamiento, especificada en un array tipo [x, y]

Este método es útil cuando se tiene una aplicación con un espacio de trabajo dinámico, y se necesita asegurarse de que la ventana aparezca en el correcto lugar en relación a otro ítem que ha sido desplegado previamente. Un útil complemento para esta característica es el

método *anchorTo*, que toma los mismos argumentos y permite a una ventana permanecer anclada a otro elemento, incluso cuando la ventana del explorador ha sido cambiada de tamaño o de desplazamiento.

Si bien muchos de los métodos de la clase *Ext.Window* simplemente proporcionan acceso a una funcionalidad existente vía código, hay muchos otros más que proporcionan avanzados escenarios que sería difícil elaborar a mano.

6.3.9 Eventos

Casi todas las acciones que hemos cubierto hasta ahora tienen sus propios eventos, que sirven para correr nuestro propio código personalizado. El evento *minimize*, es uno de los que hemos mencionado antes, porque se debe manejar manualmente este evento si se desea que el icono de la ventana realice algo. Idealmente, se espera que la ventana se almacene en una especie de área estilo taskbar para después restaurarla.

```
var w = new Ext.Window(
  {
    height: 50,
    width: 100,
    minimizable: true
  });
  w.on('minimize', doMin);
  w.show();
```

En el ejemplo de arriba, estamos creando una nueva ventana, cuya propiedad *minimizable*, se coloca en true, y entonces añadimos un evento para que el minimizado funcione, para a continuación mostrar la ventana en la pantalla. La función que manejará este evento es algo así:

```
function doMin()
  {
    w.collapse(false);
    w.alignTo(document.body, 'bl-bl');
  }
```

Nosotros simplemente le decimos a la ventana que se colapse en la parte de abajo mostrando solo la barra del título (pasando un parámetro buleano con valor false que simplemente indica que no debe animar la ventana) y entonces usamos el método *alignTo*, discutido previamente. Con los parámetros que hemos elegido, la ventana se alineará en la parte inferior izquierda del cuerpo del documento, tal como lo hiciera una ventana única en una barra de tareas.

Por supuesto, si se tuviera más ventanas, terminaríamos con una superposición en pila en la parte inferior izquierda; lo cuál no es ideal en aplicación del mundo real. Sin embargo, el ejemplo muestra como se debe manejar el evento de minimizado, y puede ser usado como una alternativa al método colapsar.

CAP.7. Toolbars, Botones y Menús

7.1 Creación de barras

7.1.1 Toolbars

Una barra de herramientas o toolbar en inglés, se puede colocar en cualquiera de los contenedores antes revisados, ya sea un panel, una ventana, un tabpanel o una forma.

Todos los contenedores tienen dos barras, una arriba y otra abajo, se pueden colocar las dos o una sola de las dos, la opción de configuración para cada una es de la siguiente forma, se coloca dentro de las propiedades de cualquier contador lo siguiente:

tbar: Para la barra de arriba (top bar)
bbar: Para la barra de abajo (bottom bar)

También se puede crear una toolbar a partir de su clase constructora de la siguiente forma:

```
new Ext.Toolbar(
  {
    renderTo: document.body,
    items: [
    {
      xtype: 'tbbutton',
      text: 'Button'
    },
    {
      xtype: 'tbbutton',
      text: 'Menu Button',
      menu: [
      {
        text: 'Better'
      },
      {
        text: 'Good'
      },
      {
        text: 'Best'
      }]
    },
    {
      xtype: 'tbsplit',
      text: 'Split Button',
      menu: [
      {
        text: 'Item One'
      },
      {
        text: 'Item Two'
      },
      {
        text: 'Item Three'
      }]
    }]
  });
```

7.1.2 Botón

La creación de un botón es bastante sencilla; la principal opción de configuración es el texto que se desplegará sobre el botón. Se puede añadir también un icono para ser utilizado junto al texto si se desea hacerlo. A continuación la sintaxis:

```
{
  xtype: 'tbbutton',
  text: 'Button'
}
```

Este botón se debe colocar en una barra superior o inferior, de la siguiente forma:

```
var w = new Ext.Window(
  {
    title: 'Mi ventana',
    height: 500,
    width: 500,
   tbar:[
   {
      xtype: 'tbbutton',
      text: 'Click me'
   }]
  });
```

7.1.3 Menú

Un menú, no es nada más que un botón con un menú desplegable, es muy simple también. Los ítems del menú trabajan con los mismos principios de los botones. Pueden tener iconos, clases y manejadores asignados a ellos. Los ítems del menú pueden también agruparse juntos para formar un conjunto de botones, pero primero se debe crear un menú estandar:

Esta es la configuración típica para un menú:

```
{
  xtype: 'tbbutton',
  text: 'Button',
  menu: [
  {
    text: 'Better'
  },
  {
    text: 'Good'
  },
  {
    text: 'Best'
  }]
}
```

Como se puede ver, una vez que la configuración del array de menús está desarrollada, los menús cobran vida. Para agrupar estos ítems de menú juntos, se necesita colocar la opción de configuración *group* con un valor igual para cada grupo que se desee colocar, además se debe colocar buleano positivo para cada item:

```
   {
      xtype: 'tbbutton',
      text: 'Button',
      menu: [
      {
        text: 'Better',
        checked: true,
        group: 'quality'
      },
      {
        text: 'Good',
        checked: false,
        group: 'quality'
      },
      {
        text: 'Best',
        checked: false,
        group: 'quality'
      }]
    }
```

7.1.4 Botón split

El botón split es un botón estándar y no es un menú combinado, con un pequeño giro. Pero usando este tipo de botón, se puede usar la funcionalidad de un botón, mientras que se añade la opción de seleccionar un item del menú adjunto. Al hacer clic en la parte izquierda del botón que contiene el texto, se activa la acción del botón. Sin embargo, al hacer clic en el lado derecho del botón, que contiene una pequeña flecha hacia abajo, se activa el menú.

```
{
   xtype: 'tbsplit',
   text: 'Split Button',
   menu: [
   {
     text: 'Item One'
   },
   {
     text: 'Item Two'
   },
   {
     text: 'Item Three'
   }]
 }
```

7.1.5 Alineación, divisores y espacios

Por defecto, los elementos de una barra de herramientas se alinean a la izquierda. No hay configuración de alineamiento para una barra, por lo tanto si se desea alinear todos los botones a la derecha, se necesita añadir un espacio lleno, como el primer elemento de una barra. Si se quiere tener ítems divididos entre ellos a la izquierda y a la derecha, también se puede utilizar un espacio lleno:

```
{
 xtype: 'tbfill'
}
```

Se debe colocar este elemento en una barra donde se desee añadir un espacio y se requiera que los ítems sean empujados hacia el final de la barra.

También se puede colocar separadores con unos pocos pixels de espacio vacío que pueden ser usados para dar espacio entre botones, o mover los elementos fuera del borde de la barra:

```
{
   xtype: 'tbspacer'
 }
```

Un divisor también puede ser colado de la misma forma:

```
{
   xtype: 'tbseparator'
 }
```

7.1.6 Accesos directos

Ext JS tiene varios accesos directos que pueden ser usados para hacer código más rápido. Los accesos directos son uno o dos caracteres que pueden ser usados en lugar de un objeto de configuración. Por ejemplo considerando un llenado estándar de una barra:

```
{
 xtype: 'tbfill'
}
```

El accesos para un llenado de una barra es un guión con un símbolo de mayor que:

'->'

No todos los accesos directos están documentados. Aquí está la lista de los accesos directos de uso común:

Componente	Acceso directo	Descripción
Llenado	'->'	El llenado o fill en inglés, es utilizado para empujar hacia la derecha los ítems de una barra
Separador	'-'	Es una barra vertical que es usada para visualizar los ítems de forma separada
Espaciador	''	Espacio en blanco usado para separar visualmente los ítems. El espacio es de dos píxeles, pero se puede cambiar reemplazando la clase CSS ytb-spacer
Texto de ítem	Cualquier texto	Agrega cualquier texto o HTML directamente en la barra de herramientas con solo colocar este texto entre comillas

7.1.7 Botones de iconos

El botón estándar puede actuar como un botón de icono o icon button en inglés, como los que se ven en los editores de texto para colocar negritas o itálicas. Se necesitan dos pasos para transformar un botón simple en un botón con icono:

- Se debe definir la imagen que será usada como icono
- Y aplicar la clase apropiada al botón

```
{
   xtype: 'tbbutton',
   cls: 'x-btn-icon',
   icon: 'imagenes/arrow.gif'
 }
```

También podríamos colocar un icono a lado del texto de un botón de la siguiente forma:

```
{
   xtype: 'tbbutton',
   cls: 'x-btn-text-icon',
   icon: 'imagenes/arrow.gif',
   text: 'Flecha'
 }
```

7.1.8 Manejadores de botones

Un botón necesita hacer más que solo lucir bonito, necesita reaccionar a una acción del usuario. Aquí es donde entran en acción los manejadores o handlers en inglés. Un *handler* es una función que es ejecutada cuando un botón o item de menú es presionado con un clic.

La configuración del *handler* es donde se añade una función:

```
{
   xtype: 'tbbutton',
   text: 'Mensaje',
   handler: function()
   {
    Ext.Msg.alert('Mensaje','Enviado desde el boton');
   }
 }
```

Este código desplegará un mensaje tipo alerta al dar clic al botón. Algunas veces se necesita que se realicen cambios al botón cuando se presiona, es así que el manejador del botón pasa una referencia a sí mismo para este propósito. El primer argumento de la función es una referencia al componente que dispara el evento.

```
{
   xtype: 'tbbutton',
   text: 'Boton',
  handler: function(f)
  {
   f.disable();
  }
 }
```

Se puede tomar esta referencia a sí mismo y acceder a todas las propiedades y funciones del botón. Por ejemplo, al llamar a la función *disable()*, el botón se transforma a color gris.

7.1.9 Cargar contenido con el clic de un botón

Vamos a hacer algo más útil en el clic de un botón. Para este ejemplo vamos a añadir una opción de configuración a cada ítem de un menú que será usado para determinar que contenido de archivo se cargará en el cuerpo de una página:

```
{
      xtype: 'tbsplit',
      text: 'Help',
      menu: [
      {
        text: 'Genre',
        helpfile: 'genre',
        handler: Movies.showHelp
      },
      {
        text: 'Director',
        helpfile: 'director',
        handler: Movies.showHelp
      },
      {
        text: 'Title',
        helpfile: 'title',
        handler: Movies.showHelp
      }]
    }
```

Nótese que tenemos una opción de configuración.

Referencia bibliográfica

- Shea Frederick, Colin Ramsay, Steve 'Cutter' Blades (2008) **Learning ExtJS**.
- Frank W. Zammetti (2009) **Practical ExtJs projects with Gears**.
- Jesus D. Garcia Jr (2009) **ExtJs in Action**.

Referencias de Internet

- Wikipedia
- http://www.desarrolloweb.com
- http://www.tierradenomadas.com/tw006.phtml