

Introducción a Sencha Touch

Índice

1 Introducción.....	2
2 Sencha Touch vs. JQuery Mobile.....	3
3 Instalar Sencha Touch.....	4
4 Estructura de carpetas.....	5
5 Código HTML básico de una aplicación.....	7
6 Instanciar una aplicación.....	9
7 Comprobando los resultados.....	10
8 Todo en un único fichero.....	12
9 Paneles y Layouts.....	12
10 Identificadores.....	14
11 Toolbars.....	15
12 Docked items.....	16
13 Botones.....	17
14 Transiciones de cambio de vista.....	21
15 TabBar.....	23
16 Carousel.....	24
17 MessageBox.....	25

1. Introducción

Sencha Touch es un framework para el desarrollo de aplicaciones móviles centrado en WebKit. Fue el primer framework basado en HTML5 y JavaScript, además utiliza CSS3 para realizar animaciones. La apariencia de las aplicaciones desarrolladas es similar al de las aplicaciones nativas en Android, BlackBerry e iOS. Sencha Touch está disponible tanto en versión con licencia comercial como con licencia Open Source GPL v3.



Una de las principales ventajas de Sencha Touch es la cantidad de controles IU o elementos de interfaz que incluye, todos ellos muy fáciles de usar y personalizar. Ha sido diseñado específicamente para dispositivos táctiles por lo que incluye una amplia gama de eventos táctiles o *gestures*, que comúnmente son usados en dispositivos táctiles. Además de los eventos estándares como *touchstart* o *touchend*, han añadido una extensa lista de eventos como *tap*, *double tap*, *tap and hold*, *swipe*, *rotate* o *drag and drop*.



Interoperabilidad con PhoneGap

Sencha Touch funciona perfectamente junto a *PhoneGap*, por lo que puede ser usado para distribuir nuestras aplicaciones en la *App Store* o en *Android Marketplace*. Se basa en el uso de un mecanismo que empotra nuestra aplicación en una shell nativa de la forma más sencilla posible. Además, gracias a *PhoneGap* podemos hacer uso de la API nativa del dispositivo para acceder a la lista de contactos, la cámara y muchas otras opciones

directamente desde JavaScript.

Integración de datos

Al igual que con *ExtJS*, Sencha Touch implementa el patrón de diseño MVC en el lado del cliente y nos ofrece una API rica y poderosa para manejar flujos de datos desde una increíble variedad de fuentes. Podemos leer datos directamente a través de AJAX, JSON, YQL o la nueva capacidad *local storage* de HTML5. Podemos enlazar esos datos a elementos específicos de nuestras vistas, y utilizar los datos sin conexión gracias a los almacenes locales.

2. Sencha Touch vs. JQuery Mobile

A continuación se enumeran las principales diferencias entre Sencha Touch y JQuery Mobile:

Sencha Touch:

- Tiene una curva de aprendizaje mucho mayor y necesita una mayor comprensión del lenguaje de programación JavaScript, pero gracias a esto proporciona una API mucho más potente.
- Dispone de un mayor número de controles para la interfaz de usuario, así como efectos de transición y animaciones entre páginas, mucho más personalizables.
- Más rápido en mayor número de dispositivos móviles (en Android a partir de la versión 2.1). El comportamiento y velocidad de Sencha Touch es mucho mejor que el de otros frameworks, a excepción de el tiempo de carga inicial, pues JQuery Mobile pesa menos.
- Al estar basado en ExtJS (usan el mismo núcleo), es muy robusto y potente, además de ser un framework muy probado y usado (también debido a que fue uno de los primeros en aparecer).
- Al igual que en ExtJS, y en comparación con JQuery Mobile, se escribe mucho. Esto podría ser tomado como un pro y como un contra. Es bueno porque indica una mayor potencia de configuración y personalización, pero por contra conlleva más tiempo de desarrollo y de aprendizaje.

JQuery Mobile:

- Muy sencillo de aprender y de implementar aplicaciones.
- Es necesario escribir muy poco código (y casi no se usa JavaScript) para lograr aplicaciones móviles muy interesantes. En lugar de orientarse a la programación JavaScript, JQuery Mobile se centra en usar etiquetas HTML con atributos definidos por el framework.
- No dispone de muchos controles para el diseño de la interfaz.
- Actualmente sigue en versión beta.
- Tiene una ejecución más lenta en los dispositivos móviles. Se espera una mejora sustancial cuando salga de fase beta.

- Al estar basado en un framework muy desarrollado, como es JQuery, funciona correctamente en un mayor número de dispositivos móviles y de navegadores, como Symbian, Android, iOS, Blackberry, Window Phone 7 o WebOS.

Ambos frameworks son buenas opciones para el desarrollo de aplicaciones móviles. Los dos utilizan HTML5, JavaScript e integran la tecnología AJAX. La decisión dependerá de las necesidades de la aplicación a desarrollar. En principio, Sencha Touch es más apropiado para aplicaciones grandes, que necesiten de mayor personalización o configuración y que vayan a hacer un mayor uso del lenguaje de programación JavaScript. JQuery Mobile se suele utilizar para aplicaciones en las que se necesite una interfaz de usuario que conecte directamente con un servidor y que haga un menor uso de JavaScript.

3. Instalar Sencha Touch

En primer lugar descargamos el SDK de Sencha Touch desde su página Web “<http://www.sencha.com/products/touch/download/>”. Tendremos que elegir si queremos usar la licencia comercial o la Open-source, pero en ambos casos el contenido a descargar será el mismo. Se nos descargará un fichero comprimido con ZIP (llamado algo como “sencha-touch-VERSION.zip”), que descomprimiremos en una carpeta de nuestro servidor Web que renombraremos a “touch”.

También es posible guardar el SDK de Sencha Touch en una carpeta diferente y posteriormente crear un enlace simbólico (llamado “touch”) hasta esta carpeta desde cada uno de nuestros proyectos Web (ver siguiente apartado).

Una vez instalado podremos comprobar su correcto funcionamiento accediendo con nuestro navegador a la dirección “<http://localhost/touch>”, con lo que veremos el contenido de ejemplo que viene con el SDK de Sencha Touch:



Sencha Touch solo funciona con navegadores basados en WebKit, como son: Safari, Google Chrome, Epiphany, Maxthon o Midori. Si lo probamos en un navegador que no lo soporte, como Firefox o Internet Explorer, solamente veremos una página en blanco o un resultado erróneo. Por lo tanto para probar nuestros proyectos Web tendremos que instalar uno de los navegadores soportados, como Google Chrome (<http://www.google.es/chrome>) o Apple Safari (<http://www.apple.com/es/safari/>).

Aunque la mayoría de webs que podemos hacer con Sencha Touch se podrían ejecutar y visualizar directamente sin necesidad de un servidor Web, sí que será necesario su uso si queremos probar nuestros proyectos utilizando algún emulador o un dispositivo móvil real.

En la sección inicial de “Instalación de un servidor Web” se puede encontrar información sobre la instalación de un emulador móvil o la configuración para el acceso externo mediante un dispositivo móvil real.

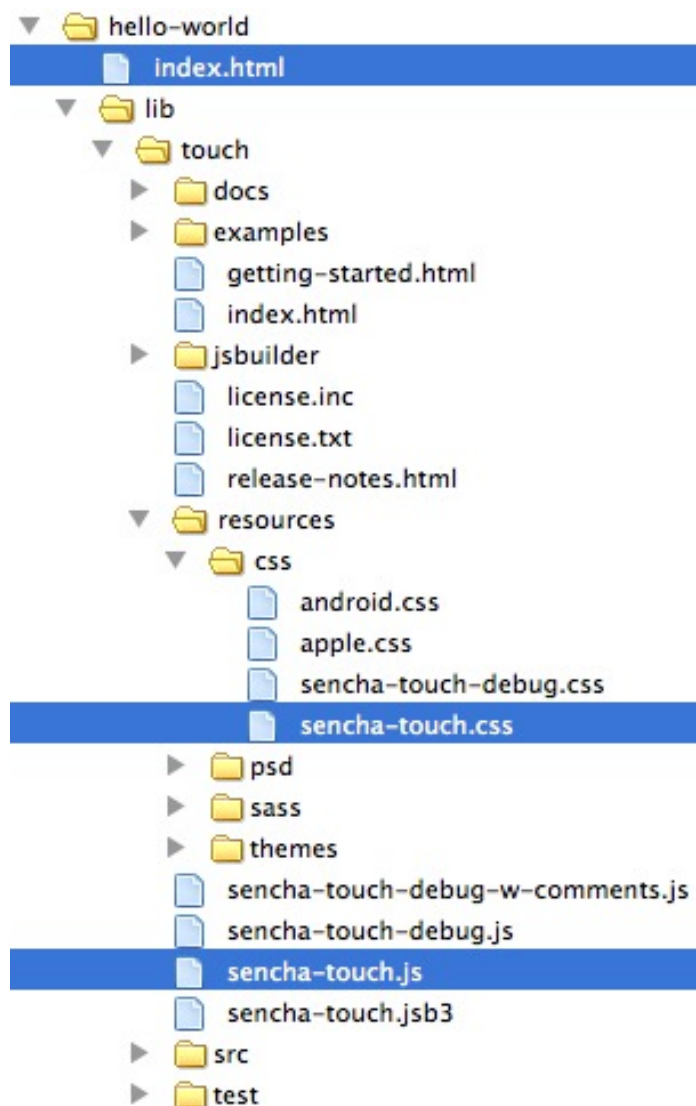
4. Estructura de carpetas

Para crear un nuevo proyecto en primer lugar tenemos que crear una carpeta dentro de nuestro servidor, por ejemplo “hello-world” como en la imagen inferior. Creamos un fichero vacío dentro de esta carpeta llamado “index.html” que será el fichero de inicio de

nuestro proyecto. Por último tenemos que copiar (o enlazar) el SDK de Sencha Touch dentro de un subdirectorio llamado “lib/touch”. Los pasos serían los siguientes:

```
$ mkdir hello-world // Nuevo proyecto
$ cd hello-world
$ touch index.html
$ mkdir lib
$ ln -s /home/code/web/SenchaSDK/sencha-touch-VERSION ./lib/touch
```

Finalmente la estructura de directorios queda como:



En la imagen se han marcado los ficheros “index.html”, “sencha-touch.css” y “sencha-touch.js”, los cuales corresponden con el fichero inicial del proyecto, la hoja de

estilo a utilizar y la librería JavaScript de Sencha Touch.

Al desplegar nuestra aplicación final no será necesario copiar todo el código de la librería de Sencha Touch, sino solamente los recursos que utilicemos.

5. Código HTML básico de una aplicación

Las aplicaciones de Sencha Touch se crean como un documento HTML5 que contiene referencias a los recursos de JavaScript y CSS. Nuestro fichero “index.html” debe de contener como mínimo el siguiente código:

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Hello World</title>

  <link href="lib/touch/resources/css/sencha-touch.css" rel="stylesheet"
        type="text/css" />

  <script src="lib/touch/sencha-touch.js" type="text/javascript"></script>
  <script src="app/app.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>
```

A continuación analizaremos por separado cada una de las partes de este código:

```
<!DOCTYPE html>
<html>
  ...
</html>
```

La primera línea nos indica que este es un documento del tipo HTML5. Las etiquetas de <html> y </html> indican el inicio y final del documento HTML y deben de contener en su interior todo el resto del código.

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Hello World</title>
  ...
</head>
<body>
</body>
```

Todo documento HTML (y HTML5 también) debe de contener primero una sección de cabecera (<head>) y a continuación una sección con el contenido principal o cuerpo del documento (<body>). En este caso el cuerpo del documento (<body>) se encuentra vacío. Esto se debe a que la librería Sencha Touch crea todo el contenido de la Web, incluidos todos los elementos de la interfaz, mediante código JavaScript.

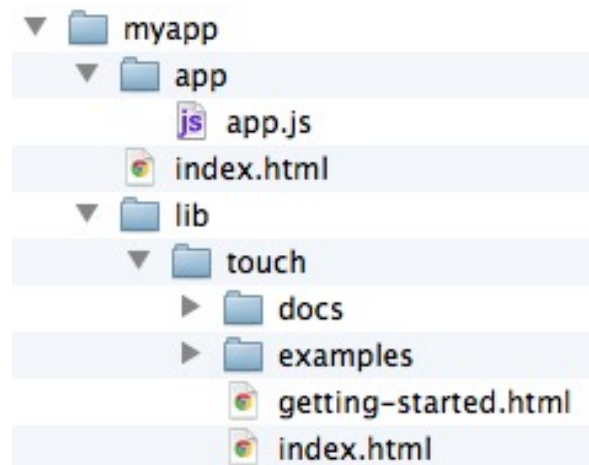
La cabecera del documento (<head>) debe de contener como mínimo los metadatos acerca del tipo de contenido, el conjunto de caracteres usados, y un título que mostrará el navegador en la parte superior de la ventana. Además debe de contener los enlaces a las librerías JavaScript y a la/s hoja/s de estilo usada/s:

```
<link href="lib/touch/resources/css/sencha-touch.css" rel="stylesheet"
type="text/css" />

<script src="lib/touch/sencha-touch.js" type="text/javascript"></script>
<script src="app/app.js" type="text/javascript"></script>
```

La etiqueta <link/> indica la localización de la hoja de estilo. Esta hoja de estilo se puede cambiar o colocar en otra dirección diferente. La etiqueta <script></script> se utiliza para cargar código JavaScript en nuestra página Web. Primero se carga la librería de Sencha Touch (localizada en “lib/touch/sencha-touch.js”) y a continuación el código de nuestra aplicación (“app/app.js”, que de momento está vacío).

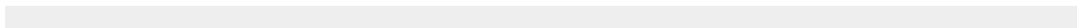
Finalmente la estructura de carpetas de nuestro proyecto Web debería quedar como el de la imagen siguiente:



Ahora ya tenemos cargadas las librerías de Sencha Touch y el código de nuestra aplicación para empezar a trabajar. De momento, si lo visualizamos en un navegador solo veremos una página en blanco.

Mostrar aviso durante la carga

Mientras que se carga la librería de Sencha Touch podemos mostrar fácilmente un texto o una imagen. Para esto podemos aprovechar el cuerpo del documento (<body>) que hasta ahora se encontraba vacío. Todo lo que incluyamos en esta sección se visualizará únicamente durante la carga, posteriormente será ocultado por el contenido de la aplicación. En el siguiente ejemplo se ha creado un cuadro centrado en el que aparece el texto "Cargando Aplicación...".




```
<body>
  <div style="margin:auto; width:220px; padding-top:100px;
font-size:16pt;">
    Cargando aplicación...
  </div>
</body>
```

6. Instanciar una aplicación

Para realizar nuestro primer ejemplo vamos a crear una aplicación que muestre en pantalla el mensaje “Hello World!”, para lo cual abriremos el fichero “app.js” y añadiremos el siguiente código:

```
var App = new Ext.Application({
  name: 'MyApp',
  useLoadMask: true,
  launch: function()
  {
    MyApp.views.viewport = new Ext.Panel({
      fullscreen: true,
      html: 'Hello World!'
    });
  }
});
```

A continuación analizaremos por separado cada una de las partes de este código:

```
var App = new Ext.Application({
  name: 'MyApp',
  launch: function()
  {
    ...
  }
});
```

Con “**new Ext.Application({ ... });**” creamos una nueva instancia de Sencha Touch, es decir, este es el constructor de nuestra aplicación. Entre las llaves “{ }” le pasaremos la lista de opciones de configuración para crear nuestra aplicación. En primer lugar le damos un nombre “**name: 'MyApp'**”, con esto automáticamente se crea una variable global llamada **MyApp** junto con los siguientes *namespaces*:

- **MyApp**
- **MyApp.views**
- **MyApp.controllers**
- **MyApp.models**
- **MyApp.stores**

Estos *namespaces* (o espacios de nombres) nos permitirán acceder a atributos de nuestra aplicación de forma sencilla, los iremos viendo en detalle más adelante.

La función “**launch: function() { }**” solo se ejecuta una vez al cargar la aplicación, y es donde deberemos de colocar el código necesario para definir nuestra interfaz de usuario.

```
MyApp.views.viewport = new Ext.Panel({  
    ...  
});
```

Con “new Ext.Panel({ ...” instanciamos un panel para nuestro contenido y se lo asignamos al “viewport” de nuestra aplicación. El “**viewport**” es la vista principal de nuestra aplicación, dentro de la cual iremos añadiendo el resto del contenido.

```
fullscreen: true,  
html: 'Hello World!'
```

Lo único que hacemos es indicarle que debe ocupar toda la pantalla (fullscreen: true) y el código HTML que tiene que contener (html: 'Hello World!'). Al poner “fullscreen: true” también activamos la opción “monitorOrientation”, para que se tengan en cuenta los eventos de cambio de orientación.

Con esto ya hemos creado nuestra primera aplicación, un panel que ocupa toda la pantalla con el texto “Hello World!”.

Nota:

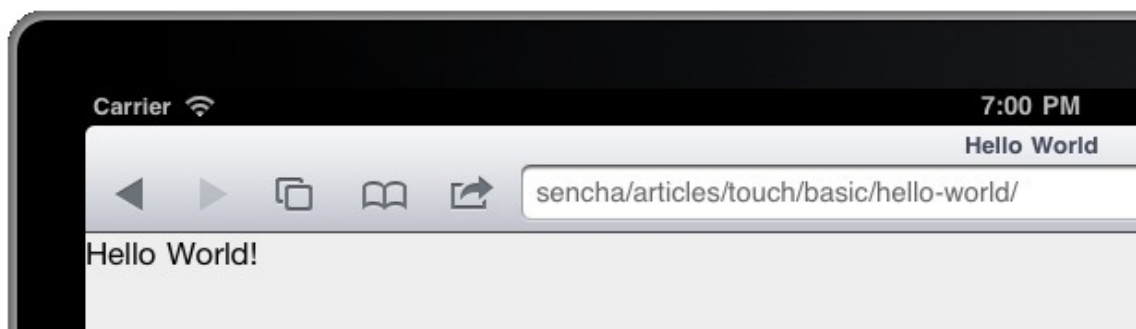
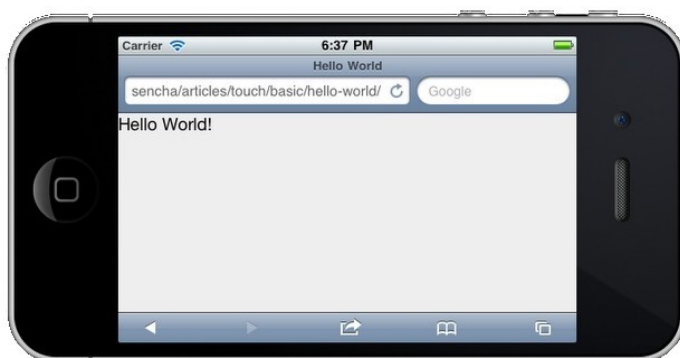
Si queremos podemos definir la función launch como una función independiente. Dentro del código del panel simplemente tendremos que poner el nombre de la función a llamar, por ejemplo: launch: crearViewport (sin poner los paréntesis de función). Y luego de forma independiente definiríamos la función function crearViewport() {} con el resto del código. Esta es una buena práctica para modularizar nuestro código.

7. Comprobando los resultados

Para comprobar el código de nuestra aplicación podemos abrirlo en un navegador compatible con WebKit, como Chrome o Safari. Si abrimos el ejemplo de la sección anterior deberíamos obtener algo como:



Si nuestro código está en un servidor también podemos comprobar el resultado utilizando un emulador de móvil, como en las imágenes inferiores:



Estos emuladores son parte del IDE de Xcode y del SDK de Android. Para poder utilizarlos necesitaremos tener nuestro código en un servidor Web. Para más información consultar la sección inicial “Instalación de un servidor Web” y Emuladores.

8. Todo en un único fichero

Para aplicaciones sencillas también es posible trabajar sin un servidor Web. Para esto enlazaremos las librerías de Sencha Touch directamente desde su SDK online (asumiendo que tenemos conexión a Internet) e incluiremos nuestro código JavaScript dentro del mismo fichero “index.html”, entre las etiquetas `<script></script>` y dentro de la sección `<head>`, como en el siguiente fragmento:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World</title>
  <link
href="http://cdn.sencha.io/touch/1.1.0/resources/css/sencha-touch.css"
                                rel="stylesheet"
type="text/css" />
  <script src="http://cdn.sencha.io/touch/1.1.0/sencha-touch.js"
type="text/javascript"></script>

  <script type="text/javascript">
    // Nuestro código JavaScript
  </script>
</head>
<body>
</body>
</html>
```

Este fichero lo podremos colocar en cualquier carpeta de nuestro ordenador y abrir directamente con un navegador. No es la mejor forma de programar, pero nos sirve para probar cosas de forma rápida con un solo fichero.

9. Paneles y Layouts

Como ya hemos comentado, en Sencha Touch el contenido se distribuye en **paneles**, los cuales actúan de contenedores de otros elementos y nos ayudan a distribuir el contenido. En primer lugar deberemos definir el panel asignado para la vista principal, el “**viewport**”, dentro del cual iremos añadiendo el resto de contenido y paneles.

```
MyApp.views.viewport = new Ext.Panel({
  fullscreen: true,
  layout: 'fit'
});
```

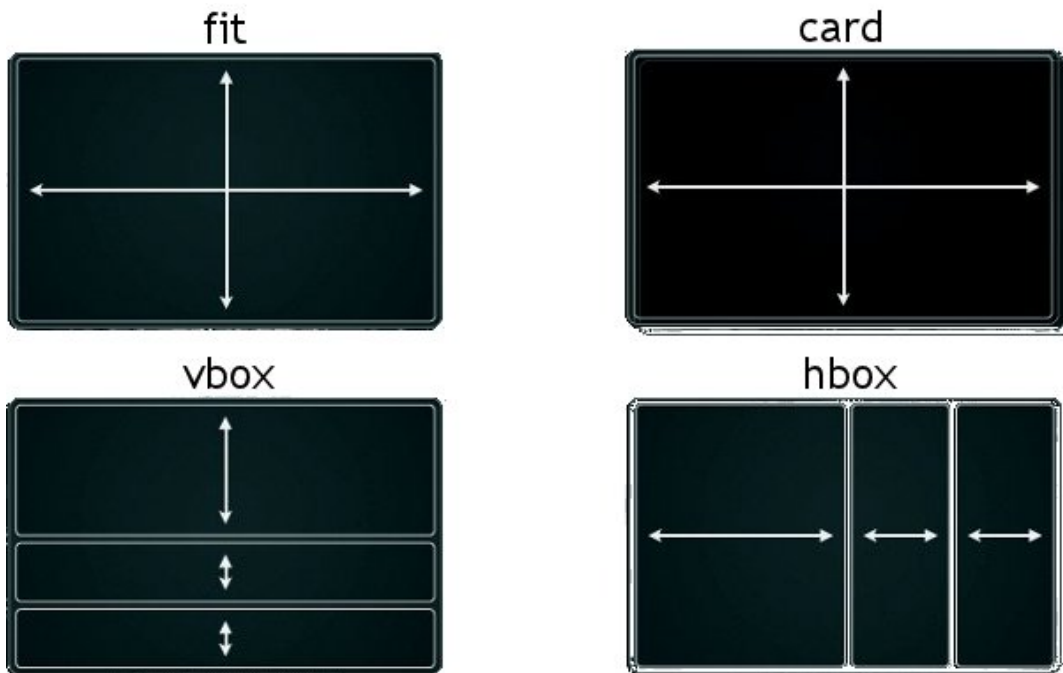
Este panel lo hemos definido para que ocupe toda la pantalla (`fullscreen: true`) y para que los elementos se distribuyan en él ocupando todo el espacio disponible (`layout: 'fit'`).

Los **layouts** se aplican a los paneles y especifican como se distribuyen los objetos que estén dentro de ellos. Los layouts que incluye Sencha Touch son:

- **fit**: los elementos se distribuyen consecutivamente ocupando todo el espacio

disponible.

- **card**: los elementos se colocan de forma superpuesta, uno encima de otro. Este elemento se utiliza para colocar paneles intercambiables, solo se visualizará uno a la vez, y el resto se mostrarán al realizar alguna acción.
- **vbox**: los elementos se distribuyen de forma vertical uno debajo de otro.
- **hbox**: los elementos se distribuyen de forma horizontal.



Con “**new Ext.Panel({ })**” creamos una instancia para un panel, dentro de este constructor se incluirán los atributos necesarios para describir su contenido. Ya hemos visto los atributos “**fullscreen: true**” y “**html: ...**”, ahora veremos que atributos tenemos que usar para definir el layout e incluir otros paneles.

Con “**layout: 'valor'**” definimos el tipo de disposición de un panel. Y usando el atributo “**items: [valores]**” podemos añadir elementos. La forma de referenciar estos elementos es a través del espacio de nombres: “**NombreAplicación.view.NombrePanel**” o a través de su identificador (como veremos más adelante). En el siguiente código se crean dos paneles (panelSuperior y panelInferior) que después se añaden al panel principal usando la disposición “vbox”:

```
MyApp.views.panelSuperior = new Ext.Panel({
    html: 'Panel Superior'
});
MyApp.views.panelInferior = new Ext.Panel({
    html: 'Panel Inferior'
});
MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
```

```

    layout: 'vbox',
    items: [
        MyApp.views.panelSuperior,
        MyApp.views.panelInferior
    ]
});

```

Items de un panel

La propiedad “items” permite añadir elementos de diferentes formas:

- Especificar un elemento o una lista de elementos utilizando su nombre:

```

items: [elemento]
items: [elemento1, elemento2]

```

- Definir un elemento o una lista de elementos en línea:

```

items: {}
items: [{...}, {...}]

```

Al crear un elemento en línea, además de especificar el resto de sus propiedades, también tendremos que definir su tipo (xtype), de la forma:

```

items: { xtype: 'toolbar', dock: 'top' }

```

En todos los casos deberemos especificar un layout apropiado.

10. Identificadores

En todos los componentes de Sencha Touch podemos definir un identificador (id) mediante el cual posteriormente podremos hacer referencia a ese elemento. La forma de definirlo, por ejemplo, para un panel es la siguiente:

```

var panelSuperior = new Ext.Panel({
    id: 'panelSuperior',
    html: 'Panel Superior'
});

```

Posteriormente desde otro elemento podremos referirnos a este panel como 'panelSuperior', por ejemplo, para añadirlo como un ítem en otro panel:

```

MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
    layout: 'card',
    items: [ 'panelSuperior' ]

    // También podríamos haber usado su nombre de variable, de la forma:
    // items: [ panelSuperior ]
});

```

Como hemos dicho, este identificador podemos usarlo con todos los elementos: botones, barras, etc. Es una buena práctica definirlo para todos los elementos que creamos que

vayamos a referenciar posteriormente. En este documento, por simplicidad, no lo incluiremos en todos los ejemplos, solamente cuando sea necesario. Cuando aparezca una referencia a otro elemento por su nombre de variable, también sería posible hacerlo por su identificador. En algunos casos solo será posible hacerlo por su nombre de identificador, como veremos más adelante.

11. Toolbars

Añadir barras de herramientas a un panel se realiza de forma muy similar a la forma de añadir paneles a otro panel. En primer lugar tenemos que crear la barra de herramientas, para esto utilizamos el constructor “**new Ext.Toolbar({ ... })**”. Dentro de este constructor incluiremos los atributos necesarios para describir su contenido.

Con “**doc: 'top'**” o “**doc: 'bottom'**” indicamos que la barra se coloque en la parte superior o en la parte inferior. Con el atributo “**title: 'texto'**” podemos indicar un texto que se colocará en el centro de la barra.

Para añadir estas barras a un panel utilizamos el atributo “**dockedItems: [...]**”, referenciando cada barra a través del espacio de nombres (“NombreAplicación.view.NombreBarra”) o a través de su identificador.

En el siguiente ejemplo se crean dos barras de herramientas (topToolbar y bottomToolbar) y después se añaden a un panel usando para una el nombre de la variable y para otra su identificador:

```
MyApp.views.topToolbar = new Ext.Toolbar({
    dock: 'top',
    title: 'Top Toolbar'
});
var bottomToolbar = new Ext.Toolbar({
    id: 'bottomToolbar',
    dock: 'bottom',
    title: 'Bottom Toolbar'
});
MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
    layout: 'fit',
    html: 'Contenido central',
    dockedItems: [
        MyApp.views.topToolbar,           // Nos referimos al elemento a través
de su variable                           // Usamos su identificador
        'bottomToolbar'
    ]
});
```

Con lo que obtendríamos un resultado similar a:



Un atributo opcional que podemos usar es “**ui: valor**” para cambiar la apariencia de la barra. Por defecto toma el valor “dark”, pero también podemos usar “light” que aplicará unos colores más claros.

12. Docked items

Como ya hemos visto en la sección anterior, **dockedItems** se utiliza para especificar uno o más elementos que serán anclados a una parte del panel. Los elementos que se suelen anclar usando esta propiedad son: **Toolbar** y **TabBar** (como veremos más adelante). Y las posiciones que podemos usar son: **top, bottom, left, right**.

Los elementos se pueden añadir definiéndolos en línea (hay que tener en cuenta que para este caso tenemos que añadir su *xtype*):

```
dockedItems: [  
  {  
    xtype: 'toolbar',  
    dock: 'top',  
    title: 'barra superior'  
  }, {  
    xtype: 'toolbar',
```

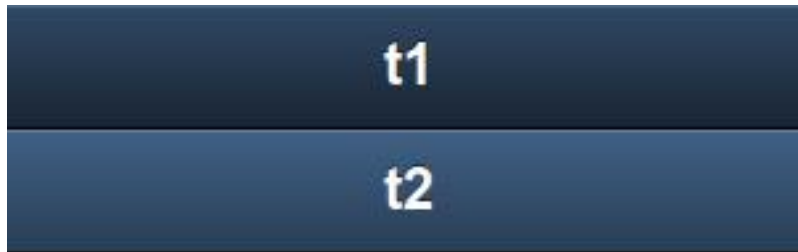


```
        dock: 'bottom',  
        title: 'barra inferior'  
    }  
];
```

Pero por cuestiones de claridad será mejor crear una definición separada y añadirlos posteriormente, como hemos visto en la sección anterior:

```
var toolbar1 = new Ext.Toolbar({ dock: 'top', title: 't1' });  
var toolbar2 = new Ext.Toolbar({ dock: 'top', title: 't2' });  
  
MyApp.views.viewport = new Ext.Panel({  
    ...  
    dockedItems: [ toolbar1, toolbar2 ]  
});
```

En este ejemplo las dos barras de herramientas se colocan en la parte superior del panel. Las barras se irán colocando una debajo de otra según el orden en el que se añaden.



También es posible manipular los elementos anclados mediante las funciones:

- `addDocked(item/array)`: ancla un nuevo elemento al objeto desde el que se referencia, por ejemplo: `panel.addDocked(bottomToolbar)`;
- `removeDocked(item)`: elimina un elemento anclado.
- `getDockedItems()`: devuelve un array con los elementos anclados.

13. Botones

Los botones se añaden a las barras de herramientas (Toolbar) mediante su propiedad “items”. Simplemente tendremos que definir el tipo de botón (ui) y el texto a mostrar (text). En el siguiente ejemplo se crea una barra de herramientas con un botón y posteriormente se añade al panel principal.

```
var topToolbar = new Ext.Toolbar({  
    dock: 'top',  
    title: 'mi barra',  
    items: [  
        { ui: 'action', text: 'Nuevo' }  
    ]  
});  
  
MyApp.views.viewport = new Ext.Panel({  
    fullscreen: true,
```

```
layout: 'fit',
dockedItems: [ topToolbar ]
});
```

Podemos usar siete tipos predefinidos de botones, estos son:

- ui: 'back'
- ui: 'forward'
- ui: 'normal'
- ui: 'round'
- ui: 'action'
- ui: 'confirm'
- ui: 'decline'



Además podemos usar los modificadores “-small” y “-round” sobre los tipos “action”, “confirm” y “decline” para obtener botones más pequeños o redondeados:



Si no seleccionamos un tipo (ui), por defecto nos aparecerá el botón tipo “normal”.



Si queremos variar el **ancho** de un botón podemos utilizar la propiedad “width: '200px'” en píxeles o “width: '95%'” indicando porcentajes.

Iconos

También podemos usar algunos iconos predefinidos, indicando el nombre del icono mediante la propiedad “iconCls: 'nombre’” y activando “iconMask: true” para que el icono se visualice correctamente, de la forma:

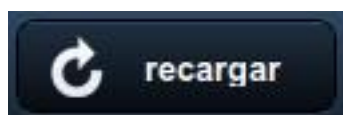
```
var button = new Ext.Button({
    iconCls: 'action',
    iconMask: true
});
```

Los iconos que podemos utilizar son:

 action	 add	 arrow_down
---	--	---

 arrow_left	 arrow_right	 arrow_up
 compose	 delete	 organize
 refresh	 reply	 search
 settings	 star	 trash
 maps	 locate	 home

Si además usamos la propiedad “text: ‘texto’” al definir el botón, el texto aparecerá a la derecha del icono:



Opcionalmente podemos aplicar colores a estos iconos, aplicándole los tipos (ui) de 'action', 'decline' o 'confirm', obteniendo:



Imágenes externas

Si queremos usar una imagen externa tenemos que aplicar al botón un estilo CSS. El botón lo definimos de forma normal, pero utilizamos su propiedad `cls` para indicarle el nombre del estilo:

```
items: [ { ui: 'normal', cls: 'btnAyuda' } ]
```

El estilo "btnAyuda" lo tendremos que definir indicando la imagen de fondo a usar junto con el ancho y el alto del botón. El tamaño de la imagen que usemos deberá de coincidir con el tamaño aplicado al botón para que no se vea recortado. El tamaño habitual de un botón es de 45x35 píxeles. Además es muy importante añadir la propiedad `!important` al cargar la imagen de fondo. Esto es debido a que Sencha Touch sobrescribe algunos estilos, y tenemos que aplicar esta propiedad para que prevalezca el nuestro:

```
.btnAyuda {
    background: url(imgs/ayuda.png) !important;
    width: 45px;
    height: 35px;
}
```

Badge

De forma muy sencilla podemos añadir una insignia distintiva a los botones para destacar alguna información. Para esto utilizamos la propiedad `“badgeText: '2’”`, que daría como resultado:



Alineaciones

Por defecto los botones salen alineados a la izquierda. Para crear otras alineaciones utilizaremos un espaciador `“{ xtype: 'spacer' }”`. En el siguiente código podemos ver diferentes ejemplos de alineaciones:

```
// Alineación derecha
items: [
    { xtype: 'spacer' },
    { ui: 'normal', text: 'Botón' }
]
// Alineación centrada
items: [
    { xtype: 'spacer' },
    { ui: 'normal', text: 'Botón' },
    { xtype: 'spacer' }
]
```

Acciones

Para añadir acciones a los botones tenemos que definir su propiedad **“handler”**, a la cual le asignaremos una función. Esta función la podemos definir en línea, de la forma `handler: function () { ... }`, o creando una función independiente para separar mejor el código, como en el ejemplo:

```
function botonPresionado(btn, evt) {
    alert("Botón '" + btn.text + "' presionado.");
}
```

```
var topToolbar = new Ext.Toolbar({
    items: [
        {
            ui: 'normal',
            text: 'Botón 1',
            handler: botonPresionado
        },
        {
            ui: 'action',
            text: 'Botón 2',
            handler: function(btn, evt) {
                alert("Botón " + btn.text + "
presionado.");
            }
        }
    ]
});
```

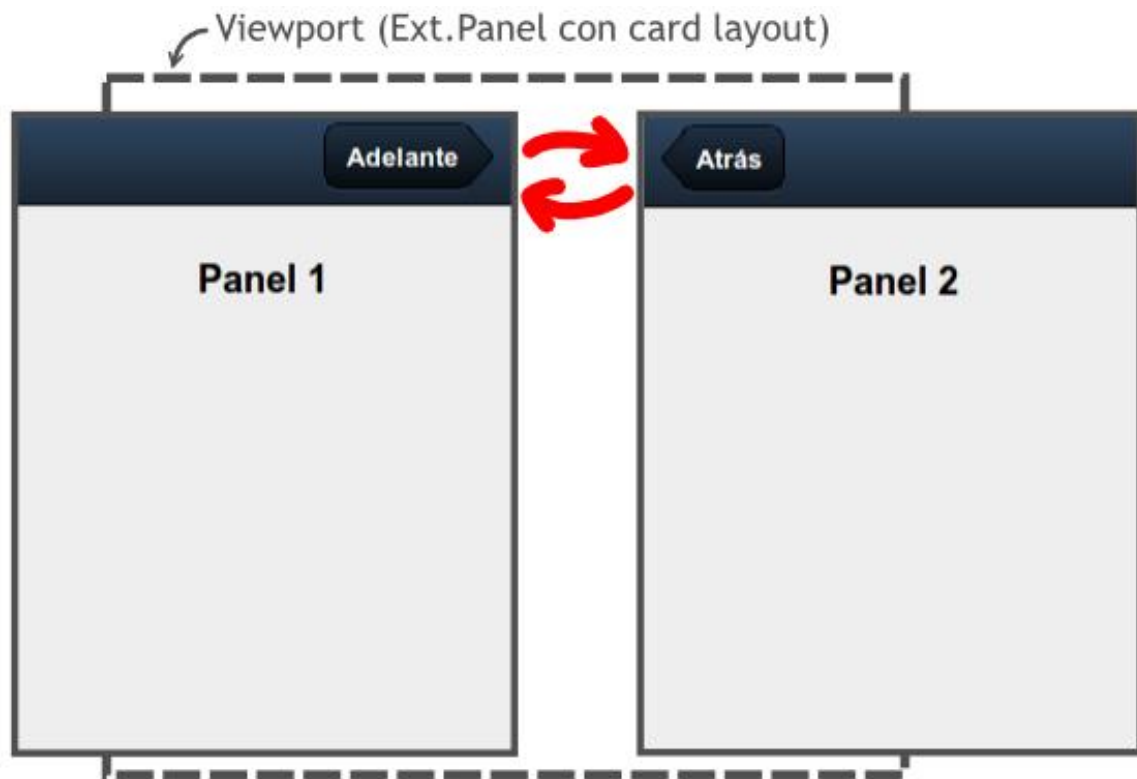
14. Transiciones de cambio de vista

En este apartado vamos a ver como cambiar entre diferentes paneles. Nuestra aplicación tendrá una vista principal y al apretar sobre algún botón, cambiaremos a una vista o panel diferente. Para hacer esto lo más importante es utilizar un panel base no visible (asignado al “viewport”), el cual contendrá como “items” los paneles entre los que queremos cambiar. Además tenemos que establecer el layout del panel base a “layout: 'card'”, quedando el código de nuestro “viewport” de la forma:

```
MyApp.views.viewport = new Ext.Panel({
    fullscreen: true,
    layout: 'card',
    items: [
        'panel1',
        'panel2'
    ]
});
```

El panel que se verá al principio es el primero que se añade a la lista de ítems, quedando el otro (o los otros) ocultos. También es muy importante fijarse que estamos **referenciando los paneles por su identificador**, y no directamente por la variable, de no hacerlo así se podrían producir comportamientos inesperados en las animaciones.

En la siguiente imagen se puede ver un esquema del intercambio de paneles (en nuestro ejemplo con dos paneles). El panel asignado al “viewport” queda como un contenedor invisible por detrás, y mediante un botón podemos pasar de un panel a otro:



A continuación se incluye el código para el “panel1”. Un simple panel con una barra de herramientas en la parte superior que contiene un botón. Lo más importante aquí es la función “handler” del botón, en la cual llamamos a `MyApp.views.viewport.setActiveItem` para cambiar al “panel2” utilizando una animación “slide”. El código para el “panel2” sería exactamente igual, pero intercambiando los identificadores.

```
var panel1 = new Ext.Panel({
    id: 'panel1',
    fullscreen: true,
    layout: 'fit',
    html: 'Panel 1',
    dockedItems: [
        {
            xtype: 'toolbar',
            items: [
                {
                    ui: 'forward',
                    text: 'Adelante',
                    handler: function()
                    {
                        MyApp.views.viewport.setActiveItem(
                            'panel2',
                            {type: 'slide', direction:
                                'up', duration: 2000});
                    }
                }
            ] // end items
        }
    ]
});
```

```
    } ] // end dockedItems
  });
```

Método setActiveItem

La función `setActiveItem(panelID, animación)` solo está definida para los paneles que tienen un layout tipo “card”. Permite cambiar entre el panel activo o visible por otro panel indicado utilizando una animación. El primer argumento de esta función es el identificador del panel que queremos colocar en primer plano. Y el segundo argumento es el tipo de animación utilizada para intercambiar los paneles. Los tipos de animaciones que podemos utilizar son:

- *fade*: difumina el panel actual, fundiéndolo con el panel de destino, hasta completar la transición.
- *pop*: realiza una especie de animación 3D. Escala el panel actual minimizándolo hasta ocultarlo, mientras que aumenta el tamaño del panel a visualizar.
- *slide*: realiza un desplazamiento para intercambiar un panel por otro, podemos indicar una dirección: left, right, up, down (por ejemplo: `direction: 'left'`).
- *flip*: realiza una animación 3D para intercambiar los paneles.
- *cube*: realiza una animación 3D para intercambiar los paneles.
- *wipe*: realiza un barrido para el intercambio. No funciona correctamente.

Para todos ellos podemos definir una duración en milisegundos (por ejemplo “`duration: 2000`”). Las animaciones “flip”, “cube” y “wipe” no funcionan correctamente en Android.

15. TabBar

Los TabBar son muy similares a las barras de herramientas, las cuales también se añaden a un panel anclándolas mediante “`dockedItems`”. La apariencia visual es diferente, con una superficie más plana (con menos sombra que las `ToolBar`). Se usan para crear botones de texto más grandes (o áreas pulsables), los cuales no tendrán aspecto de botón sino que formarán parte de la barra. Por esta razón los botones no admiten la propiedad “`ui`”, pero sí que la podemos usar para cambiar la apariencia de la barra (`ui: 'dark'` o `ui: 'light'`). Además estas barras tampoco soportan el atributo “`title`”.

En el siguiente ejemplo se ha añadido un TabBar en la parte de abajo de un ToolBar (solo se incluye el código del TabBar):

```
var bar = new Ext.TabBar({
  dock : 'top',
  ui   : 'light',
  items: [
    { text: '1st Button' },
    { text: '2nd Button' }
  ]
});
```

Con lo que obtendríamos un resultado similar al de la siguiente imagen:



También podemos usar la propiedad “**width**” para definir el ancho de los “tabs” en píxeles o en porcentaje, por ejemplo: “width: '50%'”.

Para añadir **acciones** a los tabs tenemos que definir su propiedad “handler”, asignándole una función, de la forma:

```
handler: function() {
    alert("Tab presionado");
}
```

16. Carousel

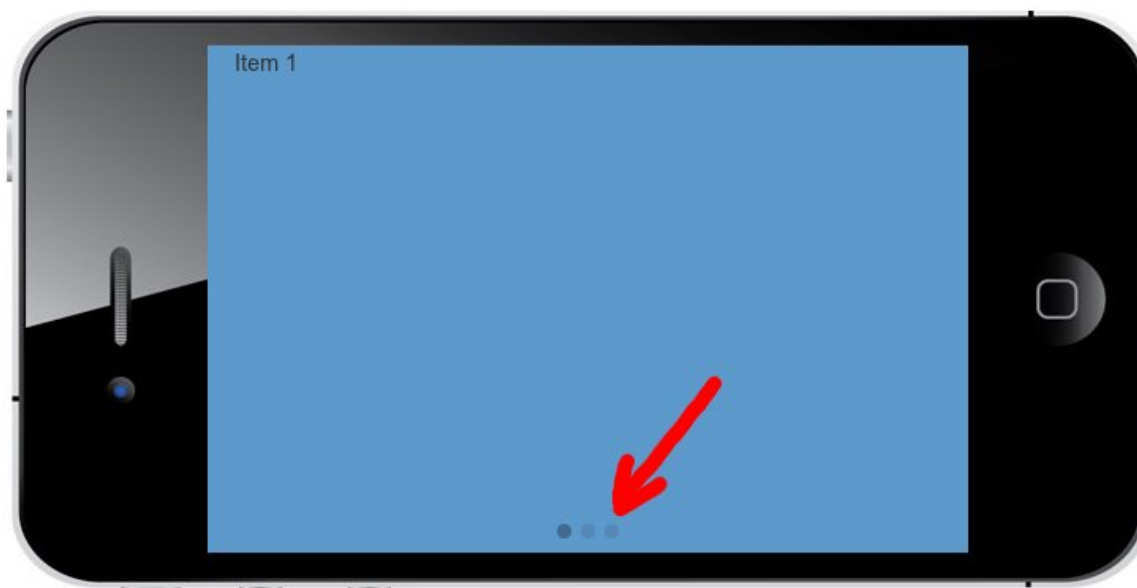
El Carousel es un contenedor de paneles que nos permite cambiar entre ellos simplemente arrastrando el dedo. Solo se muestra un panel en cada momento, además de un pequeño indicador con puntos que muestra el número de paneles disponibles. Es muy sencillo configurarlo, en la sección `items` tenemos que definir cada uno de los paneles, además si queremos que se utilicen los estilos HTML básicos tenemos que activar la opción `defaults: { styleHtmlContent: true }`, como en el siguiente ejemplo:

```
var panelCarousel = new Ext.Carousel({
    fullscreen: true,

    defaults: {
        styleHtmlContent: true
    },

    items: [
        {
            html : 'Item 1',
            style: 'background-color: #5E99CC'
        },
        {
            html : 'Item 2',
            style: 'background-color: #759E60'
        },
        {
            html : 'Item 3'
        }
    ]
});
```

Con lo que obtendríamos un resultado como el siguiente:



Una opción interesante de configuración es la orientación del panel, que básicamente lo que hace es cambiar la posición de los puntos y la dirección de movimiento de los paneles. Para configurarlo usamos la propiedad `direction: 'vertical'` (por defecto) o `direction: 'horizontal'`.

17. MessageBox

Esta clase nos permite generar mensajes emergentes de tres tipos: alertas, confirmación y de campo de texto:

ALERTAS

Muestra un mensaje de aviso con un solo botón OK, como podemos ver en la imagen siguiente



Para crear una ventana de aviso usamos el constructor `Ext.Msg.alert()`, el primer parámetro es el título de la ventana, el segundo parámetro es el mensaje de aviso que

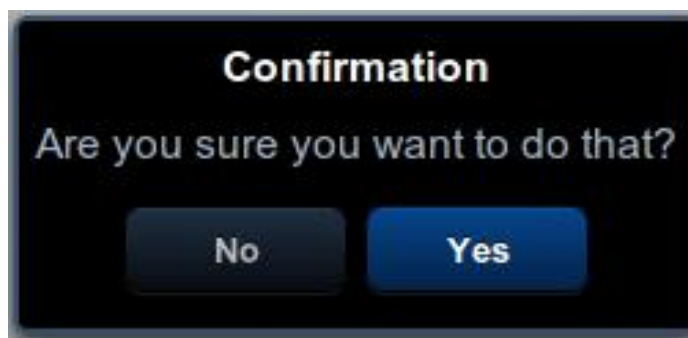
aparecerá en el centro de la ventana, y el último es la función *callback* que se llamará una vez cerrada la ventana.

```
Ext.Msg.alert('Titulo', 'Mensaje de aviso', Ext.emptyFn);
```

En este caso usamos la función vacía `Ext.emptyFn` para que no se ejecute nada. En su lugar podríamos haber puesto directamente el nombre de una función.

CONFIRM

Este mensaje de aviso nos da la opción de aceptar o rechazar, como podemos ver en la siguiente imagen:



En este caso utilizamos el constructor `Ext.Msg.confirm()`, los parámetros serán los mismos: título, mensaje y función. En este caso sí que nos interesa indicar el nombre de una función para poder comprobar si se ha pulsado el OK. La función *callback* recibirá un único parámetro cuyo valor será el texto del botón pulsado.

```
function myFunction(btn)
{
    if( btn == "yes" )
        alert( "Confirmed!" );
}

Ext.Msg.confirm( "Confirmation", "Are you sure you want to do that?",
myFunction );
```

PROMPT

El mensaje de campo de texto sirve para solicitar un dato al usuario, consiste en una pequeña ventana con un campo de texto que se puede aceptar o rechazar:



Utilizamos el constructor `Ext.Msg.prompt()` con los parámetros: título, mensaje y función. En este caso la función *callback* recibirá dos parámetros, el botón pulsado y el texto introducido.

```
Ext.Msg.prompt('Name', 'Please enter your name:', function(btn, text)
{
    alert( btn + ' ' + text );
});
```

