

Ext JS



Manual Tutorial Quizzpot.com

RESUMEN	15
INTRODUCCIÓN	16
Sobre el autor	16
Objetivo	16
Importancia de Ext	16
Ejemplos con Ext	16
INSTALACIÓN	16
¿Dónde conseguir Ext JS? ¿Hay que pagar? ¿es gratis?.....	16
Descarga	16
Licencias.....	16
INSTALACIÓN BÁSICA DEL FRAMEWORK	17
Servidor Web	17
Copiar los archivos y carpetas necesarias	17
Importar el Framework en un HTML	17
Probando el Framework.....	17
Ejecutando el ejemplo.....	18
INSTALACIÓN DE TEMAS O “LOOK AND FEEL”	18
Material de apoyo.....	18
Estilo por defecto	18
Instalación de temas	18
Descargar e instalar temas de usuarios	18
QUE SON LOS ADAPTERS Y EN QUE ME BENEFICIA.....	18
Librerías	18
Instalación.....	19
Pruebas	19
Conclusiones	19
DEFINICIÓN DE LA IMAGEN SPACER Y SU IMPORTANCIA	19
Introducción.....	19
Material de apoyo.....	19
¿Imagen externa?	20
Definiendo la ruta en nuestro servidor.....	20
Conclusiones	20
QUIZZ: PRIMER CAPÍTULO DEL CURSO DE EXT JS	20
LOCALIZACIÓN.....	22
Instalando tu idioma	22
Material de apoyo.....	22
Ejecutar el ejemplo	22
Instalar otros idiomas	22
Probar los cambios.....	22
¿Y si no existe traducción a mi lenguaje?	22
Conclusión.....	22

VARIOS IDIOMAS EN UN MISMO SITIO	22
Material de apoyo.....	23
Detectando el idioma del explorador	23
Cambiar el idioma mediante el combo.....	23
Conclusión.....	24
JAVASCRIPT ORIENTADO A OBJETOS	24
Referencias en JavaScript.....	24
Definición.....	24
Referencias sólo a objetos.....	24
Concatenación	25
¿QUE ES EL SCOPE O ALCANCE DE UNA VARIABLE?.....	25
¿QUÉ SON LOS CLOSURES?	26
Ocultar variables globales	26
Closures y Scope.....	27
Conclusiones	27
EL CONTEXTO Y LA VARIABLE “THIS”	27
El contexto	27
Cambiar el contexto de una función.....	28
Conclusiones	29
CREACIÓN DE OBJETOS Y MÉTODOS PÚBLICOS.....	29
Creación de objetos	30
Métodos públicos.....	30
Conclusiones	31
MÉTODOS PRIVADOS Y MÉTODOS PRIVILEGIADOS	31
Propiedades y métodos privados	31
Métodos privilegiados.....	31
Conclusiones	33
PAQUETES Y NAMESPACE	33
Conclusiones	34
HERENCIA EN JAVASCRIPT	34
Conclusión.....	35
QUIZZ: CONCEPTOS FUNDAMENTALES DE JAVASCRIPT (PARTE I)	35
QUIZZ: JAVASCRIPT ORIENTADO A OBJETOS (PARTE II)	38
CONCEPTOS DE EXTJS	40
¿QUÉ ES EL “XTYPE”?	40
Definición.....	40
¿Cuál es la manera “lazy”?	40

¿Cómo puedo mejorar el rendimiento de mi aplicación?	40
Ventajas de utilizar la configuración xtype	40
Creación de nuestro propio “ xtype”	41
Conclusiones	41
TRABAJANDO CON EL DOM	41
CONOCIENDO AL OBJETO “ELEMENT”	41
Esperar cuando el DOM esté listo para usarse	41
Obtener un elemento.....	41
Mejorado el rendimiento	42
Conclusiones	42
LA CLASE “DOMHELPER”.....	42
Material de apoyo.....	43
Definiendo el namespace del tutorial.....	43
Crear elementos.....	43
Aplicar estilos.....	43
Insertar elementos.....	44
Conclusiones	44
Material de apoyo.....	44
Ext.DomQuery.....	44
Paso 1: Empaquetando el componente	44
Paso 2: Dando estilo a los tabs	44
Paso 3: Desplegando el tab inicial	45
Paso 4: Asignar estilos al div contenedor	45
Paso 5: Crear el evento onClick	45
Paso 6: Definiendo el Clic	45
Paso 7: Seleccionando el tab correcto	46
Paso 8: Mostrar el contenido seleccionado	46
Conclusiones	46
APLICAR EFECTOS VISUALES A ELEMENTOS.....	46
Empaquetando el tutorial	47
Fading	47
Frame.....	47
Ghost	47
Highlight.....	48
Puff	48
Scale.....	48
Slide	48
Shift	48
Posiciones	49
Easing.....	49
Conclusiones	49
QUIZZ: TRABAJANDO CON EL DOM.....	50
MENSAJES Y EVENTOS	52
MENSAJES Y ALERTAS AL USUARIO	52
Material de apoyo.....	52

Empaquetando el ejemplo del tutorial.....	52
Alertas	52
Confirmación.....	52
Prompt.....	53
Wait	53
Callbacks	53
Mensajes personalizados	54
Conclusiones	55
MANEJO DE EVENTOS SOBRE ELEMENTOS	55
El problema	55
La solución	55
Componentes.....	56
Conclusiones	56
QUIZZ: MENSAJES Y EVENTOS	56
AJAX Y EL OBJETO STORE.....	58
El objeto Ajax, peticiones GET y POST	58
Material de apoyo.....	58
Namespace	58
Crear los eventos	58
Ajax.....	59
La función success	60
La función failure	60
El servidor	60
Conclusiones	61
¿QUÉ ES UN STORE Y CÓMO FUNCIONA?.....	62
Material de apoyo.....	62
Encapsulando el tutorial.....	62
La información	62
Crear un Store con información local	63
Cargar la información en el Store	63
Crear los “listeners” de los botones	63
Ordenar los registros.....	64
Filtrar registros en el store	64
Buscar registros.....	65
Buscar por una propiedad	65
Contar los registros del store	66
El Log.....	67
Conclusiones	67
LEER INFORMACIÓN DE UN XML	67
Material de apoyo.....	67
Estructura general.....	67
El XML a utilizar.....	70
Paso 1: Crear el registro Person.....	71
Paso 2: Crear el “reader” para XML.....	72
Paso 3: Crear el Proxy.....	72
Paso 4: Crear el Store	72
Paso 5: Cargar la información en el Store	72
Conclusiones	73

LEER INFORMACIÓN EN FORMATO JSON.....	73
Material de apoyo.....	73
La información	73
Creando el Store	74
El resto del código	75
Conclusiones	75
 QUIZZ: AJAX Y EL OBJETO STORE.....	76
 PANELES, VENTANAS Y PESTAÑAS	79
 ¿QUE ES UN PANEL, PARA QUE SIRVE Y CÓMO SE CREAN?.....	79
Material de apoyo.....	79
¿Qué es un panel?	79
Crear un primer panel	79
El contenido del panel..	80
Colapsar los paneles.....	81
La propiedad “defaults”	83
Cargar la información	84
Conclusiones	84
 UNA VENTANA FLOTANTE.....	85
Material de apoyo.....	85
Empaquetando.....	85
Una ventana básica	85
Otras configuraciones	86
Minimizar una ventana	86
Contenido	87
Cargando sitios externos	87
Conclusiones	88
 BARRAS DE HERRAMIENTAS EN PANELES Y VENTANAS	88
Material de apoyo.....	88
Ext JS 3.0	88
Empaquetando el tutorial	88
Barra de herramientas	89
Agregar íconos a los botones.....	90
Botones alineados a la derecha	93
Una caja de texto en la barra de herramientas.....	95
Agrupando botones	95
“Split buttons” y menús	96
Asignar acciones a los botones.....	97
Conclusiones	99
 BARRA DE STATUS EN VENTANAS Y PANELES	99
Material de apoyo.....	99
Empaquetando el tutorial	99
Crear la barra de estado.....	100
Texto en el Statusbar	100
Programando un reloj	101
Contador de palabras.....	102
Auto guardado	103

Conclusiones	104
LAS PESTAÑAS O TABS.....	104
Material de apoyo.....	104
El TabPanel	104
Agregar Tabs en tiempo de ejecución.....	106
Agregar un scroll a los tabs.....	106
LOS FORMULARIOS	108
FORMULARIOS Y CAMPOS COMUNES EN EXT JS.....	108
Material de apoyo.....	108
Empaquetando el tutorial	108
Formulario	108
Campos de texto	109
Checkbox.....	110
Radiobuttons	112
Campos ocultos.....	113
Botones en el formulario.....	114
Ventana que alojara el formulario	116
Conclusiones	117
COMBO BOX CARGADOS LOCAL Y REMOTAMENTE (AJAX).....	117
Material de apoyo.....	117
Empaquetando el tutorial.	117
Ventana	118
ComboBox Local.....	118
ComboBox Remoto	120
Datos con formato	122
Una variación del ComboBox	123
Conclusiones	124
COMBOS DEPENDIENTES EN EXTJS	124
Material de apoyo.....	124
La fuente de información	124
Creación de los JsonStore.....	125
Creación de los ComboBox dependientes	126
Desplegarlos en una ventana	127
Agregando el “listener” adecuado.....	127
Conclusión.....	128
UN CALENDARIO PARA CAPTURAR FECHAS	128
Material de apoyo.....	128
Empaquetando el tutorial.	128
Ventana	129
DateField Simple	129
Un DateField Restringido	130
Conclusiones	131
LLENAR FORMULARIOS CON INFORMACIÓN DEL SERVIDOR.....	131
Material de Apoyo	131
Demostración.....	131

Información	132
Solicitar la información al servidor	132
Crear la lista del “Top Ten”	133
Agregar “listeners” a las imágenes	134
Crear el formulario	134
Conclusiones	136
GUARDAR INFORMACIÓN EN EL SERVIDOR	136
Material de apoyo	136
Demostración	136
Submit tradicional	137
Hacer el submit con el botón save	137
Hacer un submit utilizando Ajax	138
Personalizar el envío	139
Parámetros extra	140
Mensaje cargando	141
Conclusiones	141
VALIDACIONES SIMPLES EN FORMULARIOS	141
Material de apoyo	142
Demostración	142
Campos requeridos	142
Máximo de caracteres en un TexField	143
Validaciones comunes	143
Sólo caracteres alpha	143
Validar un correo electrónico (e-mail)	143
Validar un URL	144
Validar en el servidor	144
Conclusiones	145
VALIDACIONES PERSONALIZADAS	145
Material de apoyo	146
Mostrar los mensajes de error	146
Validar mayoría de edad	146
Validar un teléfono	147
Validar una tarjeta de crédito	148
Conclusiones	149
CAMPOS COMPUESTOS	149
La base de datos	149
Empaquetando el tutorial	150
Creando el formulario	150
Función compositeFields	151
Enviando la información	152
Guardado la información	152
Mensaje de éxito	153
Conclusión	154
GRÁFICAS	154
GRÁFICAS EN EXT JS 3	154
Material de apoyo	154

Empaquetar el tutorial	154
Definir la información a graficar	154
Las gráficas.....	154
Gráfica de barras.....	155
Gráfica lineal	155
Gráfica de pastel (Pie Chart).....	156
Colocarlas en pantalla	156
Conclusiones	158
GRAFICANDO SERIES	158
Material de apoyo.....	158
Definir la información a graficar	158
Creación de la gráfica	159
Agregando series.....	159
Agregar leyenda	160
Cambiar el texto de la leyenda	161
Formato a los ejes	162
Conclusiones	163
LAS TABLAS.....	164
UNA TABLA BÁSICA CON INFORMACIÓN CARGADA DE UN ARRAY	164
Empaquetando el tutorial	164
El Store.....	164
La Tabla	165
El ColumnModel.....	166
El SelectionModel	167
Conclusiones	168
MOSTRAR INFORMACIÓN DE UN ARCHIVO XML.....	168
Material de apoyo.....	168
Empaquetando el código.....	168
El XML a utilizar.....	168
El registro “Person”	169
Crear el “Reader”	169
Crear el Store y cargar la información	170
Crear el Grid	170
Crear la ventana	170
Conclusiones	171
INFORMACIÓN CONTENIDA EN FORMATO JSON.....	172
Material de apoyo.....	172
Empaquetando el tutorial	172
El JSON a utilizar	173
Crear el “Record”	173
Crear el “Reader”	173
Crear el Store y cargar la información	174
Ahormando algunas líneas de código.....	174
Crear el Grid	174
Desplegar el grid	174
Conclusiones	175

PAGINADO REMOTO EN UN GRID.....	175
Material de apoyo.....	176
Definir el “namespace”	176
El paginado	176
Crear el Store	177
Crear el Grid	177
Crear el PaggingToolbar	178
Asignar la barra de paginación al Grid	178
Enviar parámetros extra.....	179
Conclusiones	181
 FORMATO A LA INFORMACIÓN DE LAS CELDAS.....	181
Material de apoyo.....	181
Información a mostrar	181
Definir el Namespace	182
Creación del Store	183
Creación del paginador	183
Crear el Grid y la Ventana contenedor	183
Cambiar los parámetros del paginador	184
Mostrar la imagen	185
Dos campos en una misma celda.....	187
Corregir la descripción.....	187
Cambiando el texto booleano	188
Conclusiones	189
 UTILIZANDO FECHAS Y TIEMPO EN UN GRID.....	189
Material de apoyo.....	189
Información a mostrar	189
Namespace	190
Crear el Store y el Grid	190
Problemas con el manejo de las fechas y tiempos.....	192
Solución al problema de ordenación	193
Cambiar el formato de las celdas	195
Conclusiones	196
 EDITAR LA FILA DE UN GRID EN UN FORMULARIO.....	196
Demostración.....	196
Material de apoyo.....	197
Doble clic en una fila	197
Obtener el record del store	198
Crear el formulario de edición.....	198
Crear la ventana contenedora del formulario	199
Llenar los campos del formulario	199
Vista previa de la imagen	200
Guardar la información del formulario.....	202
Conclusiones	202
 GRID CON COLUMNAS DINÁMICAS.....	202
La base de datos.....	203
Exponer la información	204
Encapsulando el código	205
Solicitando la información al servidor	205
Función “createGrid”	205
Función “error”	207

Conclusión.....	207
EDICIÓN DE TABLAS	208
EDITAR LAS CELDAS DE UN GRID.....	208
Demostración.....	208
Material de apoyo.....	208
El servidor	208
Definiendo el “JsonStore”	209
Crear el Grid.....	209
Mostrar la ventana contenedora.....	210
Cargar la información en el “Grid”	210
Mostrar una caja de texto al dar doble clic.....	211
Filtrar solamente números.....	212
Campos sucios.....	212
Navegación con el teclado.....	213
Conclusiones	213
EDITAR UNA CELDA MEDIANTE UN COMBOBOX.....	213
Demostración.....	213
Material de apoyo.....	214
Información del Grid	214
Crear el Grid editable	215
Crear la ventana contenedora	215
Crear el combo para seleccionar los años.....	216
Crear un combo dinámico	217
Desplegar el género y no el identificador	219
Cambiando el “scope”	220
Asegurarse que siempre exista contenido en el store del combo	221
Conclusiones	221
GUARDAR LOS CAMBIOS DEL GRID EDITABLE USANDO AJAX	221
Material de apoyo.....	221
Demostración.....	221
Guardar la información en el servidor	222
Creación del Grid editable	223
Guardar los registros modificados.....	224
Crear un registro nuevo.....	227
Cancelar los cambios.....	229
Conclusiones	230
AGRUPADO Y SUMATORIA DE COLUMNAS	231
La base de datos.....	231
Exponer la información	232
Preparando el Entorno	233
Empaquetando el Tutorial.....	233
Preparando la Información para el GRID	234
Creando el GRID	234
Group Summary	237
Guardando la Información	239
Conclusión.....	241

CRUD DE UN CATÁLOGO DE CONTACTOS.....	242
La Base de Datos	242
Exponer la Información	243
Empaquetando el Tutorial.....	244
Creado el CRUD	244
Creando un Grid Editable	246
Grabar los Registro Modificados	247
Crear un registro nuevo.....	249
Eliminar un registro.....	251
Conclusión.....	252
Árboles con ExtJS	252
El tree panel.....	252
Demo	252
Material de apoyo.....	252
La información a desplegar	253
Creación del TreePanel.....	254
Expandiendo los nodos	254
Conclusiones	255
 ÁRBOLES GENERADOS CON AJAX.....	 255
Demostración.....	255
Material de apoyo.....	255
Crear el TreePanel.....	255
Mostrando el árbol	256
La información en el servidor	257
Conclusiones	258
 CAMBIAR EL ÍCONO A LOS NODOS DE UN TREEPANEL.....	 258
Material de apoyo.....	259
Namespace	259
Creación del TreePanel.....	259
Mostrar el Tree en una ventana	260
Definiendo el organigrama.....	260
Cambiando los íconos	261
Conclusiones	264
 DESPLEGAR EL SISTEMA DE ARCHIVOS EN UN TREEPANEL.....	 264
Material de apoyo.....	265
Leyendo el contenido de una carpeta.....	265
Creación del TreePanel.....	267
Conclusiones	268
 EXPLORADOR DE ARCHIVOS PARA EL SERVIDOR.....	 268
Material de apoyo.....	269
Creando el Layout	269
Crear las regiones.....	270
Permitir que se redimensione el TreePanel.....	271
Desplegando el contenido de los archivos.....	272
Conclusión.....	274
 ORDENAR LOS NODOS DE UN TREEPANEL	 274
La demostración.....	274
Material de apoyo.....	275

Creación de la base de datos	275
Creación de la estructura de árbol	276
Generar el JSON a desplegar	278
Creación del TreePanel.....	279
Guardando el orden	281
Conclusiones	284
LAYOUTS Y MAQUETACIÓN.....	285
BORDER LAYOUT	285
Material de apoyo.....	285
Definición de un layout de tipo border.....	286
Crear márgenes entre las regiones	289
Redimensionando las regiones.....	290
Border Layout dentro de una región	292
Conclusiones	293
PLUGINS Y EXTENSIONES.....	294
HERENCIA CON EXTJS.....	294
Crear prototipos a la manera antigua.....	294
Crear instancias del prototipo	295
Sobre escritura de métodos	296
Conclusiones	297
EJEMPLOS VARIADOS	298
INTEGRACIÓN DEL FORMPANEL Y EL GRIDPANEL	298
La Base de datos.....	298
Exponer la información	300
Empaquetando el Tutorial.....	301
Creando el CRUD	301
Creado el Grid	302
Creado el Formulario.....	303
Grabar los Registros Modificados.....	305
Crear un Registro Nuevo	308
Eliminando un Registro	310
Conclusión.....	311
CREANDO UN WIZARD	312
Empaquetando el Tutorial.....	312
Creando el Cardlayout.....	313
Definición de los pasos en el Wizard	313
Función “createAllias”	313
Función “createDriver”	315
Función “createCredentials”	318
Avanzar en los pasos del wizard	319
Retroceder en los pasos del wizard	320
Finalizando el wizard.....	320
Conclusion.....	321

EDITANDO UNA IMAGEN	321
Empaquetando el Tutorial.....	321
Colocando la Imagen	321
Creando los Sliders.....	323
Interactuando con la Imagen.....	324
Conclusión.....	325
UNA ENCUESTA DINÁMICA	325
La base de datos.....	325
Exponer la información	327
Encapsulando el Código	328
Solicitando la Información desde el Servidor vía Ajax	328
Creando el Layout	328
Creando la Encuesta.....	329
Guardando en nuestro Servidor	331
Actualizando los resultados en la base de datos.....	332
Mensaje de éxito.....	333
Gráfica con Resultados.....	333
Conclusiones	335
INTEGRACIÓN DEL TREEPANEL Y TABPANEL	336
Encapsulando el código	336
Creando el Layout	336
Creación del menú principal.....	337
Pestaña inicial	339
Asignar eventos al menú principal	339
Conclusiones	340
EXTJS DESKTOP – CREACIÓN DE MÓDULOS	341
Iniciar el Desktop	341
Modificar las referencias	341
Importar nuestro módulo.....	341
Agregando nuestro módulo al Desktop	341
Regitrando el módulo en el Desktop	342

Resumen

Este documento es un pequeño compilado de una exelente librería de gestión de aplicación esta información fue sacada desde la <http://www.quizzpot.com>, siendo esta institución una opción a desarrollarse de manera profesional en este tipo de framework.

Comparado con otras herramientas como jquery.ui es mucha mas practico al momento de implementar o ejecutar funciones y la creación de aplicaciones es mas dynamica..

Con pocas líneas de código es posible realizar interfaces amigables para el desarrollo rápido de aplicaciones con un look and feel totalmente novedoso.

Introducción

Presentación del ponente, definición de objetivos y alcance del curso, también se hacen delimitaciones, limitaciones y requisitos para tomar el curso.

En este capítulo se plantea el alcance del curso, se muestran algunos ejemplos de lo que el alumno será capaz de hacer y la importancia de conocer esta herramienta.

Sobre el autor

Mi nombre es Crysfel Villa, soy ingeniero de software, me gusta el desarrollo y la implementación de nuevas tecnologías, estaré impartiendo este curso pero si alguien más desea apoyar realizando algunos temas, es bienvenido.

Objetivo

El principal objetivo es comenzar con temas sencillos hasta llegar a un nivel de complejidad como el desarrollo de extensiones y plugins. La meta es que el curso no tarde más de 4 meses apartir de hoy, por lo tanto los temas serán publicados frecuentemente.

Importancia de Ext

En la actualidad es importante tomar en cuenta el aspecto de las pantallas de los sistemas que desarrollamos, ya que el usuario es la parte más importante en nuestras aplicaciones. Ext JS nos ayuda a mejorar la experiencia del usuario, pues nos proporciona componentes con funcionalidades avanzadas y de fácil implementación.

Ejemplos con Ext

Los siguientes links son aplicaciones realizadas con Ext JS, puedes visitarlas y verlas por ti mismo.

- [Mensajes y alertas](#), es muy común mandarle mensajes al usuario, Ext JS nos proporciona diferentes tipos de mensajes.
- [Ventanas](#), las ventanas son muy fáciles de realizar y pueden ser muy útiles en nuestras aplicaciones.
- [Grid](#), aquí se muestra la funcionalidad de un grid editable.
- [Feed viewer](#), esta aplicación es capaz de administrar feeds (rss)
- [Escritorio web](#), este ejemplo muestra la forma en que se puede realizar una aplicación tipo escritorio, el ejemplo es muy interesante.

Instalación

Conocimientos y conceptos básicos para comenzar a trabajar a tu manera, es importante saberlos para evitar problemas en el futuro.

¿Dónde conseguir Ext JS? ¿Hay que pagar? ¿es gratis?

En este primer capítulo se menciona donde descargar el código fuente del framework, se habla acerca de las licencias existentes y sus diferencias.

Descarga

La página oficial es www.extjs.com, desde allí se puede descargar la versión oficial del framework. En el menú Productos -> Descargar se encuentran tres tipos de descarga:

- Descarga completa: contiene el código fuente, el build, ejemplos y ayuda.
- La documentación: se puede descargar el API para futuras referencias, es conveniente hacerlo para poder utilizarla en nuestra propia máquina.
- Descarga personalizada: este tipo de descarga permite seleccionar los componentes específicos del framework, de esta manera se puede optimizar la librería.

Licencias

Existen tres tipos de licencias, cada una tiene sus ventajas y desventajas, además de que algunas tienen restricciones que debes saber.

- Licencia comercial: esta licencia la debes comprar cuando necesites desarrollar software propietario. El precio va desde \$289 a \$14,449 dólares dependiendo de la cantidad de desarrolladores.

- Licencia open source: este tipo de licencia aplica cuando deseas desarrollar un proyecto open source, esto implica liberar tu proyecto con licencia GNU GPL V3.
- Licencia revendedor: este tipo de licencia es necesaria adquirirla cuando deseas realizar un framework o librería basada sobre Ext JS. Para saber el costo es necesario ponerse en contacto con el equipo de Ext JS.

Instalación básica del Framework

Una vez que sabemos donde descargar la librería oficial de Ext JS, lo siguiente es aprender como instalar el framework en el proyecto que vamos a comenzar, en este tema se explica como hacerlo.

Servidor Web

Para este tema y la mayor parte del curso, voy a usar un servidor Web. Para no tener problemas con la instalación voy a utilizar un sencillo instalador de Apache2, PHP5, MySQL; el instalador se llama AppServ y lo puedes [descargar desde su sitio oficial](#). La instalación es muy sencilla solo sigue los pasos del Wizard, una vez instalado es necesario abrir el directorio donde lo instalaste y en la carpeta “www” crea un directorio que se llame “curso”, dentro de este, crea otro directorio que se llame “ext-2.2” y otro que se llame “instalacion” (sin el acento).

Copiar los archivos y carpetas necesarias

Dentro del directorio ext-2.2 es necesario copiar los siguientes archivos y carpetas de la librería que descargamos en el tema anterior.

- ext-all.js: en este archivo se encuentran todos los widgets y componentes del framework.
- adapter: en esta carpeta se encuentran varias librerías como prototype, jQuery, YUI y Ext JS, en otro tema se explica detalladamente el uso de estos archivos, por ahora simplemente copia todo el contenido.
- Resources: en esta carpeta se encuentran las imágenes y estilos necesarios para los componentes del framework.

Importar el Framework en un HTML

Para importar el Framework a un documento HTML, es necesario importar el estilo, el adapter y los componentes. A continuación se muestran las respectivas rutas:

- ext-2.2/resources/css/ext-all.css
- ext-2.2/adapter/ext/ext-base.js
- ext-2.2/ext-all.js

En la parte superior derecha de esta página puedes descargar el HTML que se utilizará para este ejemplo, también puedes descargar el código fuente o ver el ejemplo ejecutándose.

Para importar los archivos mencionados lo hacemos de la siguiente manera:

1. <link rel="stylesheet" type="text/css" href="../ext-2.2/resources/css/ext-all.css" />
2. <script type="text/javascript" src="../ext-2.2/adapter/ext/ext-base.js"></script>
3. <script type="text/javascript" src="../ext-2.2/ext-all.js"></script>

Probando el Framework

Para probar la instalación, voy a mandar mostrar un alert cuando el DOM este listo para ser utilizado, así que voy a poner un listener al evento “onReady”, de la siguiente manera:

1. Ext.onReady(function(){
2. alert('well done!!');
3. });

El evento onReady es muy semejante al evento onLoad, la diferencia es que el evento onLoad se dispara hasta que todos los elementos de la página han sido completamente descargados al navegador; esto incluye imágenes, estilos, JavaScript y el DOM construido. El evento onReady se dispara cuando el DOM está listo, no importa que las imágenes no han terminado de descargarse completamente; esto nos ayuda a no tener que esperar tiempos largos para poder ejecutar nuestro código.

Ejecutando el ejemplo

Para ejecutar lo que acabamos de realizar es necesario ir a tu explorador favorito y abrir el HTML que acabamos de hacer. En la URL escribe `http://localhost`, con esto accedemos al directorio raíz del servidor que instalamos al principio, ahí debe aparecer la carpeta curso; dentro de esta debe estar la carpeta “ext-2.2” e “instalacion”, en la carpeta instalación es donde se guardó el HTML. Si todo ha salido bien, al abrir el HTML aparecerá el alert mostrando un mensaje, el cual nos indica que instalamos correctamente el Framework.

Instalación de temas o “look and feel”

Por defecto, los componentes de Ext JS vienen en color azul claro, el cual es llamativo pero quizás no del gusto de todos; es por esto que existen diferentes estilos los cuales se pueden instalar y cambiar la apariencia de los componentes.

Material de apoyo

Para este tema es necesario descargar el material de apoyo que se encuentra en la parte superior derecha de la pantalla, es necesario descomprimir el zip y copiar el HTML dentro de la carpeta “instalacion” que hemos creado en temas anteriores. Recuerda que esa carpeta se encuentra dentro del servidor Web que se instaló al principio del curso.

Estilo por defecto

Al ejecutar el HTML descargado desde el explorador (ya sea Firefox, IE, etc) aparecerá una ventana de color azul claro; este es el estilo por defecto de los componentes Ext.

Instalación de temas

Junto con la librería que se descargó en el principio viene otro tema en color gris (`ext/resources/css/xtheme-gray.css`), es necesario importar este archivo al documento HTML en el que estamos trabajando agregándolo después del archivo “`ext-all.css`”, de la siguiente manera:

1. `<link rel="stylesheet" type="text/css" href="../ext-2.2/resources/css/ext-all.css" />`
2. `<link rel="stylesheet" type="text/css" href="../ext-2.2/resources/css/xtheme-gray.css" />`

Al actualizar la página de ejemplo podemos ver que la ventana ha cambiado de aspecto, ahora se ve en color gris.

Descargar e instalar temas de usuarios

La comunidad de usuarios de Ext ha creado varios temas más, estos se pueden descargar desde [la página oficial](#).

Una vez que descargas el tema de tu agrado descomprime el ZIP; aparecen dos carpetas, una que se llama “css” y otra que se llama “images”. Para instalar el tema es necesario copiar el contenido de la carpeta “css” e “images” a la carpeta donde se encuentra el framework de Ext en `ext-2.2/resources/css` y `ext-2.2/resources/images` respectivamente. Después de copiar las carpetas debes importar el estilo al HTML de la siguiente manera:

1. `<link rel="stylesheet" type="text/css" href="../ext-2.2/resources/css/ext-all.css" />`
2. `<link rel="stylesheet" type="text/css" href="../ext-2.2/resources/css/xtheme-slickness.css" />`

De esta manera habrá sido instalado el nuevo tema, actualizamos la página de ejemplo y veremos el cambio de aspecto de la venta.

Que son los adapters y en que me beneficia

El patrón Adapter (Adaptador) se utiliza para adecuar una interfaz de tal forma que pueda ser utilizada por una clase que de otro modo no se podría utilizar.

Librerías

Gracias a que Ext JS nació como una extensión para la librería YUI y debido a la buena estructura del framework, ahora podemos ejecutar Ext JS sobre otras librerías.

Las implementaciones que existen en la actualidad de este patrón (adapter) son para la librería de jQuery, Prototype y YUI, además Ext JS cuenta con su propia librería la cual se llama Ext-base. Los adapters los encontramos en el directorio ext-2.2/adapter.

Instalación

Para instalar la librería deseada simplemente es necesario importarla al documento, inmediatamente después importar el adapter y por ultimo importar el framework de Ext JS, a continuación un ejemplo de cómo instalar jQuery.

1. <script type="text/javascript" src="../ext-2.2/adapter/jquery/jquery.js"></script>
2. <script type="text/javascript" src="../ext-2.2/adapter/jquery/ext-jquery-adapter.js"></script>
3. <script type="text/javascript" src="../ext-2.2/ext-all.js"></script>

Pruebas

Para probar que se ha instalado correctamente la librería de jQuery, es necesario escribir código de jQuery para que sea ejecutado. Podemos probar la instalación de jQuery al cambiar el evento onReady de Ext JS al estilo jQuery.

```
1. $(function(){  
2.     $(document.body).css('background-color','#ccc');  
3.     var win = new Ext.Window({  
4.         title:'Hello world',  
5.         width:400,  
6.         height:300,  
7.         minimizable:true,  
8.         maximizable:true  
9.     });  
10.    win.show();  
11.});
```

Otra prueba más que se puede hacer es cambiar el color de fondo del body. Esto se puede realizar con jQuery fácilmente, solo basta con agregar esta línea dentro del evento ready.

1. \$(document.body).css('background-color','#333');

Conclusiones

Como se ha mostrado, es muy sencillo instalar la librería que más nos guste, además de que Ext JS se comporta igual con cualquiera de éstas. Durante el resto del curso utilizaremos Ext-base, que es la librería propia de Ext JS, pero cada quién puede utilizar la librería de su agrado o la que mejor conozca.

Definición de la imagen spacer y su importancia

Los componentes de Ext JS utilizan una imagen GIF de 1px por 1px en color transparente para generar espacio y colocar iconos o separadores, es necesario saber esto para evitar problemas en el futuro.

Introducción

Es importante saber sobre la imagen que Ext JS utiliza para generar espacios, pues algunas veces los componentes no se visualizarán correctamente si no tenemos conexión a Internet; esto ocurre por que este GIF está alojado fuera de nuestro servidor, en este tema hablaré de cómo solucionar esto.

Material de apoyo

Antes de continuar es necesario descargar el material de apoyo en la parte superior derecha de la pantalla, una vez descargado descomprime el ZIP y copia el HTML dentro de la carpeta "instalacion" que creamos al principio del curso. Lo siguiente es ir al explorador y ejecutar el ejemplo. Al abrir el HTML en el explorador aparece una ventana que contiene una estructura de árbol en su interior, este ejemplo es muy sencillo, simplemente muestra esos componentes que no tienen funcionalidad alguna.

¿Imagen externa?

Si abrimos el Firebug en la pestaña de NET, podremos ver los recursos que se están usando para generar lo que vemos en pantalla; por ejemplo el ext-base.js, el ext-all.js y algunas otras imágenes, todas estas alojadas en nuestro servidor local, pero hay una imagen que se está cargando desde www.extjs.com, ésta es la imagen de la que estamos hablando en este tema! La pregunta aquí es, ¿por qué no cargarla desde mi servidor local? La respuesta es sencilla, no se está cargado de nuestro servidor local por que ¿cómo sabría el Framework de Ext JS donde está? Por eso se está cargando desde otro servidor donde él SI sabe que está.

Definiendo la ruta en nuestro servidor

En ocasiones nuestros sistemas o aplicaciones no tienen acceso a Internet, en este caso los componentes no podrán desplegarse correctamente debido a que no se puede descargar la imagen "s.gif". Para solucionar esto es necesario indicarle al Framework donde encontrar esta imagen en nuestro servidor.

La imagen "s.gif" se encuentra dentro de la carpeta misma de Ext JS (ext-2.2). La ruta es **ext-2.2\resources\images\default\s.gif**, esta ruta se la debemos colocar dentro del HTML que descargamos (spacer.html) de la siguiente manera:

1. Ext.BLANK_IMAGE_URL = '../ext-2.2/resources/images/default/s.gif';

BLANK_IMAGE_URL es una constante que contiene la ruta a la imagen spacer, esta línea de código se debe colocar antes de comenzar a crear los componentes, es decir, se debe poner al principio del documento.

Conclusiones

Este tema es muy básico, pero he conocido a muchos desarrolladores que no saben acerca de esto, es por eso que me parece importante mencionarlo en el primer capítulo de este curso. Otro punto a tomar en cuenta es que al definir la ruta en nuestro servidor la descarga es mucho más rápida que descargar la imagen desde extjs.com, con esto lograremos que los componentes se desplieguen más rápido.

Quizz: Primer capítulo del curso de Ext JS

El primer capítulo del curso de Ext JS ha terminado, es momento de hacer una pequeña evaluación de lo aprendido.

1.- ¿Cuáles son los tipos de licencias que tiene Ext JS?

- Creative Commons Attribution.
- Commercial, GNU GPL, Reseller.
- Open source.
- Shareware.

2.- ¿Es posible descargar una versión personalizada de Ext JS?

- Si
- No
- No sé

3.- ¿Cuales son los archivos básicos que se deben importar al documento HTML para que EXT JS funcione correctamente?

- ext.js, ext.css
- jQuery.js, ext.js, ext-all.css

- ext-all.js, ext-base.js, ext-all.css
- ext-base.js

4.- ¿Qué es el evento onReady?

- Es igual al evento onLoad.
- Es un evento que se ejecuta cuando el DOM está listo.
- Es un evento para monitorear llamadas AJAX al servidor.
- Ninguna de las anteriores

5.- ¿Cómo se instalan nuevos temas o skins para los componentes?

- Sólo es necesario copiar las imágenes y los estilos a la carpeta resources.
- Importando el estilo (css) del tema antes del ext-all.css
- No se pueden instalar más temas, Ext JS sólo cuenta con un tema que no se puede cambiar.
- Importando el estilo (css) del tema justo después del ext-all.css

6.- ¿Para que sirve el patrón Adapter?

- Para adecuar una interfaz de tal forma que pueda ser utilizada por una clase que de otro modo no se podría utilizar.
- Para evitar que se acceda a los métodos y propiedades de la clase que implementa.
- Para adaptar una fachada.
- Para nada bueno.

7.- ¿Cuáles son las librerías que Ext JS puede utilizar para ejecutarse por medio de un Adapter?

- Mootools, prototype, jQuery.
- Prototype, jQuery, YUI.
- XPath, FX, jQuery.
- YUI, Scriptaculous, Dojo.

8.- ¿Por qué llaman los componentes de Ext JS a una imagen externa a nuestro servidor, la cual está alojada en www.extjs.com?

- Para que no desperdiciemos ancho de banda
- Por que no saben la ruta exacta en nuestro servidor
- Para tener mas visitas.
- No sé.

9.- ¿En dónde se define la ruta a la imagen “spacer”?

- Ext.BLANK_IMAGE_URL
- Ext.IMAGE_SPACER
- Ext.CONTEXT_PATH
- Ext.BLANK_IMAGE_PATH

Localización

Mediante ExtJS, es posible internacionalizar los widgets en diferentes lenguajes, por defecto se utiliza el Inglés, pero es posible hacer traducciones a otros lenguajes o buscar si alguien mas ya la ha realizado.

Instalando tu idioma

Por defecto los componentes de Ext JS utilizan el idioma Inglés para desplegar textos y mensajes, pero es posible cambiarlo por el que necesitemos.

Material de apoyo

Antes de continuar asegúrate de descargar el material de apoyo de este tema. Crea una carpeta en el directorio “curso” (la carpeta donde hemos estado trabajando en este curso ubicada en el servidor Web) que se llame “lenguaje”, dentro de esta descomprime el material de apoyo para que funcione correctamente.

Ejecutar el ejemplo

Vamos a ejecutar en el explorador el material de apoyo, al hacerlo aparece una ventana que contiene un grid con información. En la parte inferior del grid aparece el mensaje “page 1 of 1”, también en la cabecera de las tablas aparece en inglés las opciones de ordenar ascendente y descendente. Al dar clic sobre el botón “Add” aparece un pequeño formulario, los 2 campos son requeridos y uno de ellos es de tipo fecha en el cual aparece un calendario con los meses en inglés, además si ingresas una fecha inválida te mandará un mensaje de error en inglés.

Instalar otros idiomas

Es sencillo instalar otros idiomas o lenguajes, primero tienes que ir a la carpeta donde descargamos el Framework en el primer tema, dentro de esta carpeta hay un directorio llamado “build”, allí aparece una carpeta llamada “locale”; se necesita copiar esta carpeta dentro de la carpeta ext-2.2 que está en el servidor Web donde estamos trabajando. Dentro de este directorio se encuentran las traducciones a varios idiomas más, lo que tenemos que hacer es buscar el que deseamos instalar e importarlo al HTML del material de apoyo inmediatamente después de importar el ext-all.js, de esta manera se sobrescriben los textos por el lenguaje que hemos importando.

1. <script type="text/javascript" src="../ext-2.2/ext-all.js"></script>
2. <!-- debemos importar el idioma después del ext-all.js -->
3. <script type="text/javascript" src="../ext-2.2/locale/ext-lang-es-min.js"></script>

Probar los cambios

Después de guardar los cambios hay que actualizar el explorador para poder ver los componentes en el idioma que instalamos anteriormente. Es importante mencionar que la traducción que hemos importado sólo afecta a los componentes de Ext JS y no a la información creada por el usuario, como es el caso de las cabeceras y los botones “add” y “remove”, pues queda de parte del programador definirlos.

¿Y si no existe traducción a mi lenguaje?

En caso de que no existiese una traducción para tu idioma dentro de la carpeta “locale”, o si por ejemplo necesitas hacer una variación de algún idioma propio de tu país, entonces lo que tienes que hacer es copiar alguno de los lenguajes existentes (en la carpeta “locale”), renombrarlo y comenzar a traducir sólo lo que se encuentra dentro de las comillas.

Conclusión

Como se ha mostrado es muy sencillo cambiar el idioma a los componentes, hacer una traducción también lo es, lo único que tenemos que tener claro es que el archivo JS con la traducción se debe importar después del ext-all.js, de esta manera se sobrescriben los textos con el idioma importado.

Varios idiomas en un mismo sitio

En ocasiones es necesario crear sitios o sistemas multilenguaje, en este tema se muestra como detectar el idioma del explorador usando PHP, además de permitir que el usuario pueda cambiar el idioma mediante un combo.

Material de apoyo

Es necesario descargar el material de apoyo, descomprimirlo y copiar los archivos al servidor Web que instalamos previamente en nuestra computadora, dentro de la carpeta “lenguaje” que se creó en el tema anterior. Al ejecutar el archivo “multilanguage.php” podemos ver que es exactamente el mismo ejemplo del tema anterior a diferencia de que ahora aparece un combo en la parte superior derecha con algunos idiomas definidos.

Detectando el idioma del explorador

Vamos a editar el archivo “multilanguage.php”. Al inicio del archivo vamos a escribir el código necesario para detectar el idioma del explorador, esto se hace leyendo el contenido de la variable SERVER de la siguiente manera:

```
1. $lang = $_SERVER['HTTP_ACCEPT_LANGUAGE'];
```

Con esa instrucción tenemos en la variable “\$lang” el idioma del explorador en formato es, en, it, fr, etc... pero si el explorador tiene configurado el idioma propio de un país el valor de la variable “\$lang” es diferente, por ejemplo el español de México sería es_MX, el español de Chile es_CL, el de Argentina es_AR, en_UK para el inglés del Reino Unido. Lo que necesitamos hacer es extraer los primeros dos caracteres de la variable “lang” de la siguiente manera:

```
1. $lang = substr($lang,0,2);
```

Por último es necesario importar al HTML el fichero con el idioma adecuado, de la siguiente manera:

```
1. <script type="text/javascript" src="../ext-2.2/locale/ext-lang-<?php echo $lang; ?>-min.js"></script>
```

Recuerda que se tiene que importar después del archivo “ext-all.js” para que sobrescriba el idioma que Ext JS tiene por defecto, una vez guardados los cambios puedes probarlos actualizando el explorador donde está el ejemplo. En el video se muestran tres diferentes exploradores con lenguajes diferentes.

Cambiar el idioma mediante el combo

Al seleccionar un lenguaje del combo la página se recarga y envía mediante la variable “lang” el lenguaje seleccionado, para este tema no vamos a prestarle atención a cómo lo hace, más adelante explicaré como agregarle eventos a elementos del documento, por ahora enfocaré este tema solo a la detección de idiomas.

Es necesario verificar si se recibe el parámetro “lang”, pues éste tendrá mayor importancia ya que el usuario ha seleccionado un lenguaje mediante el combo. Si en el request viene el parámetro “lang”, entonces importaremos el lenguaje de la traducción solicitada de lo contrario importaremos el idioma del explorador, esto se traduce a código PHP de la siguiente manera:

```
1. if(!isset($_GET['lang'])){  
2.     $lang = $_SERVER['HTTP_ACCEPT_LANGUAGE'];  
3.     //es, es_MX, en, en_UK  
4.     $lang = substr($lang,0,2);  
5. }else{  
6.     //Si el usuario ha seleccionado un lenguaje del combo  
7. }
```

El siguiente paso es tomar el parámetro “lang” y validar si contiene un lenguaje válido, en este caso vamos a verificar que contenga alguno de los cinco idiomas que soportará el ejemplo, para realizar esto utilizaré un bloque switch.

```
1. $lang = $_GET['lang'];  
2. switch($lang){  
3.     case 'en':  
4.     case 'es':
```

```

5.     case 'it':
6.     case 'pt':
7.     case 'ro':
8.         break;
9.     default:
10.        $lang = 'en';
11.    }

```

Si la variable “lang” contiene en, es, ro, pt o it significa que el contenido de la variable es correcto, por lo tanto no es necesario hacer nada más, pero si no contiene ninguno de estos valores significa que hay un error y le asignamos un valor por defecto, en este caso sería inglés (en).

Conclusión

Esta actividad que acabamos de terminar es suficiente para realizar sitios multilenguaje, como puedes ver fue muy sencillo. Si tienes alguna duda o sugerencia puedes dejarla en los comentarios.

JavaScript orientado a objetos

Se definen términos y conceptos para programar orientado a objetos en JavaScript utilizando algunas utilerías que ExtJS nos proporciona para una fácil implementación.

Referencias en JavaScript

Una referencia es un puntero al lugar exacto donde se encuentra un objeto, en JavaScript este es un concepto fundamental que debemos conocer y dominar.

Definición

Físicamente los objetos están alojados en la memoria y accedemos a ellos mediante una referencia, la cual esta contenida en una variable. Múltiples variables pueden hacer referencia al mismo objeto, a su vez este objeto puede contener referencias a otros objetos como por ejemplo strings, numbers, arrays, etc. Cuando múltiples variables apuntan a un mismo objeto y éste es modificado, el cambio se reflejará en todas las variables que están haciendo referencia al objeto. Un ejemplo de esto es lo siguiente:

```

1. //Se crea un objeto vacío
2. var obj = {};
3.
4. //Se crea una referencia
5. var reference = obj;
6.
7. //Se agrega una propiedad al objeto original
8. obj.property = 1;
9.
10. //La referencia puede acceder a la propiedad recién creada
11. console.debug('reference.property = '+reference.property);

```

El mismo principio se aplica para los arreglos, aunque estos se modifican a si mismos mediante el método “push” las referencias se verán afectadas. Analicemos el ejemplo siguiente:

```

1. var array = ['Ext JS','Mootools','jQuery'];
2. var ref = array;
3.
4. array.push('prototype');
5.
6. console.debug(ref.length == array.length);
7. console.debug(ref);

```

Referencias sólo a objetos

Es importante mencionar que en JavaScript las referencias sólo apuntan a objetos en memoria y no a otras referencias como el lenguaje C/C++. En el siguiente ejemplo se muestra este comportamiento:

```

1. //Se crea el objeto original
2. var obj1 = {property:'Original value'};
3. //Se hace una referencia al objeto original
4. var ref1 = obj1;
5.
6. //obj1 apunta a un Nuevo objeto
7. obj1 = {property:'New Object!'};
8.
9. //Ref1 apunta al objeto original, por lo tanto son diferentes
10. console.debug('same object = '+ (obj1.property == ref1.property));
11. console.debug(obj1.property);
12. console.debug(ref1.property);

```

Concatenación

Los strings también son objetos y hacemos referencia a estos mediante una variable; es importante recordar que cuando se concatena uno o más strings siempre resulta un nuevo objeto. En el siguiente ejemplo se muestra que al concatenar un texto se crea un nuevo string y por lo tanto la referencia queda apuntando al string original.

```

1. var str = 'Hello world!';
2. var refStr = str;
3.
4. str += ' this is Crysfel';
5. console.debug('same string = '+ (str === refStr));

```

Si tienen alguna duda o sugerencia sobre este tema de referencias, pueden hacer preguntas en los comentarios, con gusto las responderé.

¿Que es el scope o alcance de una variable?

Cuando nos referimos al “scope” nos estamos refiriendo al alcance o al área donde una variable puede utilizarse, en este tema veremos como maneja JavaScript este concepto.

En JavaScript el alcance de las variables sucede de una función y no de bloques (if, while, switch, etc.) como en lenguajes Java o C/C++; en otras palabras si se define una variable dentro de un bloque condicional (if) esta variable se podrá utilizar en toda la función en cuestión y no solamente en el bloque definido. Veamos el siguiente ejemplo que demuestra lo que se acaba de mencionar:

```

1. if(true){
2.     var test = 'is it a block var?';
3. }
4.
5. function testing(){
6.     var test = 'testing scope!';
7. }
8. testing();
9.
10. console.debug(test);

```

Dentro de la condición se ha definido la variable “test”, en lenguajes como Java ésta variable debería existir sólo dentro de la condición, pero en JavaScript no sucede de esta manera ya que esa variable ha sido definida en el “global scope” y no dentro del bloque condicional. Por otro lado la variable que se definió dentro de la función “testing” sólo existe dentro de esa función.

Es importante mencionar que las variables declaradas en el “global scope” son propiedades del objeto “window”, para comprobar esta afirmación basta con hacer lo siguiente:

```

1. var global = 'this is a global var!';
2. console.debug(window.global);

```

Otro punto a tomar en cuenta es que cuando no se declaran las variables utilizando la palabra reservada “var” no importa si están dentro de una función o no, estas variables automáticamente serán definidas en el “global scope”.

```
1. function globalScopeFunction(){  
2.     globalScope = 'this is a new var in the global scope!';  
3. }  
4. globalScopeFunction();  
5.  
6. console.debug(globalScope);  
7. console.debug(window.globalScope);
```

Es importante saber estos conceptos pues los necesitaremos más adelante, además de que ahora sabremos con exactitud el alcance de las variables y donde podemos usarlas.

¿Qué son los Closures?

Mediante los closures podemos resolver varios problemas cuando desarrollamos componentes o alguna aplicación, es importante conocer este concepto aplicado a JavaScript.

Un closure es la manera en como una función dentro de otra función contenedora puede hacer referencia a las variables después de que la función contenedora ha terminado de ejecutarse. Este concepto puede ser un poco difícil de comprender y de explicar así que veamos un ejemplo.

```
1. function sayHi(seconds){  
2.     var hi = 'Hi folks!';  
3.  
4.     setTimeout(function(){  
5.         console.info(hi); //Referenciando a la variable 'hi'  
6.     },seconds*1000);  
7. }  
8.  
9. sayHi(2);
```

En el ejemplo anterior se puede ver claramente como la función “sayHi” termina su ejecución y después de 1 segundo se ejecuta la función interna mostrando el mensaje contenido en la variable “hi”, la cual pertenece a la función contenedora, a esto se le llama closure.

Ocultar variables globales

Muchas veces declaramos variables en el “global scope”, es una mala práctica hacer esto por que estas variables pueden interferir con algunas librerías o con el código de algún otro miembro de nuestro equipo. Si utilizamos una función anónima auto ejecutable y hacemos uso de closures podemos resolver de una manera sencilla este problema. A continuación se muestra como hacer esto:

```
1. (function(args){  
2.     var thisWasGlobal = 'closure!';  
3.  
4.     window.onload = function(){  
5.         console.info(thisWasGlobal);  
6.     }  
7.  
8. })();
```

El código anterior encapsula las variables declaradas dentro de la función anónima, de esta manera las variables estarán en un scope donde no hay peligro que sobre escriban a otras variables.

Closures y Scope

Se ha mostrado que un closure permite referenciar variables que existen y pertenecen a la función contenedora. Es importante mencionar que al hacer un closure, éste toma el último valor de la variable de la función contenedora. Un caso muy común es cuando utilizamos un ciclo o loop.

```
1. window.onload = function(){
2.     var el = document.getElementById('element');
3.     var events = ['click','mouseover','mouseout'];
4.
5.     for(var i=0;i<events.length;i++){
6.         var item = events[i];
7.         el['on'+item] = function(){
8.             console.info('event: '+item);
9.         }
10.    }
11. }
```

Al ejecutar el código anterior se agregan los eventos contenidos en el arreglo “events” al elemento seleccionado, el problema se ocasiona cuando se imprime en la consola el evento que se ejecuta pues siempre imprime lo mismo, en este caso “mouseout”, esto sucede porque la variable “item” contiene a “mouseout” como último valor asignado. Para solucionar este problema es necesario crear un scope diferente para cada iteración del ciclo, de esta manera se crearán variables diferentes; esto se realiza mediante una función anónima que se auto ejecute. Si tienes alguna duda en cuanto al concepto scope te recomiendo ver el tema anterior. El código quedaría de la siguiente manera.

```
1. window.onload = function(){
2.     var el = document.getElementById('element');
3.     var events = ['click','mouseover','mouseout'];
4.
5.     for(var i=0;i<events.length;i++){
6.         (function(){ //función anónima crea un nuevo scope
7.             var item = events[i]; //item pertenece a la function anónima
8.             el['on'+item] = function(){
9.                 console.info('event: '+item); //un closure de la función anónima
10.            }
11.        })();
12.    }
13. }
```

Si ejecutamos el código anterior veremos que ahora el evento correcto se imprime en la consola.

Conclusiones

En este tema vimos que es un closure y cuán útil es cuando introducimos el concepto de scope, esto es algo que debemos aprender y tener en cuenta a la hora de desarrollar nuestros proyectos. El tema de closures es complicado, te recomiendo darle una leída a “JavaScript Closures” escrito por Jim Jey, es un excelente material que debes leer.

El contexto y la variable “this”

El contexto es el objeto en el que se está operando al tiempo de ejecución, en JavaScript tu código siempre tendrá un contexto. En este tema se muestra como se puede cambiar o asignar el contexto a métodos o funciones.

El contexto

La manera como funciona el contexto es mediante la variable “this” la cual hace referencia al objeto en el que está contenido el código en ejecución. En temas anteriores se ha mostrado que el “global scope” está contenido en el objeto “window” por lo tanto si se imprime en la consola de Firebug, la variable “this” desplegará al objeto “window”.

```

1. // this == window
2. console.debug(this);
3.
4. function test(){
5.     // this == window
6.     console.debug(this);
7. }
8.
9. test();

```

La función “test” ha creado un nuevo scope pero no se le ha especificado un contexto en cual debe ejecutarse por lo tanto toma el contexto global y la variable “this” dentro del scope de la función “test” apunta al objeto “window”, es por eso que al imprimir en la consola la variable “this” mostrará al objeto “window”. Ahora veamos un ejemplo de una función en el contexto de un objeto:

```

1. var obj = {
2.     name: 'obj',
3.     run: function(){
4.         // this == obj
5.         this.value = 1;
6.         console.debug(this.name);
7.     }
8. };
9.
10. obj.run();

```

En el código anterior se muestra cómo es que mediante la palabra reservada “this” se hace referencia al objeto “obj”, de esta manera se le pueden agregar o leer propiedades al objeto en cuestión.

Cambiar el contexto de una función

Cuando se crea una función anónima o una función dentro de algún método del objeto “obj” adopta un nuevo contexto por lo tanto estas funciones están en el contexto global, examinemos el siguiente ejemplo:

```

1. var obj = {
2.     name: 'obj',
3.     run: function(){
4.         this.value = 1;
5.         console.debug(this); // this == obj
6.
7.         (function(){ // se crea un nuevo scope
8.             console.debug(this); // this == window
9.         })();
10.
11.         function test(){ // se crea un nuevo scope
12.             console.debug(this); // this == window
13.         }
14.
15.         test();
16.     }
17. };
18.
19. obj.run();

```

Como se puede ver la variable “this” dentro de la función anónima y la función “test” apuntan al objeto “window” en lugar de apuntar al objeto “obj”, esto sucede porque estas funciones no han sido definidas como métodos el objeto “obj”, por lo tanto no pertenecen a “obj” y adoptan el contexto global. A continuación se muestra el mismo código anterior con una modificación en la función anónima y en la función “test”:

```

var obj = {
1.     name: 'obj',
2.     run: function(){
3.         this.value = 1;
4.         console.debug(this); // this == obj
5.
6.         (function(){
7.             console.debug(this); // this == obj
8.         }).call(this); // se autoejecuta con el método call
9.
10.    this.test = function(){ // se define como método de obj
11.        console.debug(this); // this == obj
12.    }
13.
14.    this.test(); // se ejecuta dentro del contexto de obj
15. }
16. };
17.
18. obj.run();

```

En la función anónima se está utilizando el método “call” para ejecutarse a si misma y se le pasa como parámetro el objeto que utilizará como contexto al momento de ser ejecutada; de esta manera hemos cambiado el contexto global que anteriormente tenía por el contexto “obj”. Para la función “test” se ha cambiado la manera en que se definía anteriormente y ahora se ha definido como un método del objeto “obj”, de esta manera se ejecutará en el mismo contexto. El llamado de la función también ha cambiado, ya que ahora le anteponemos la variable “this” al método “test”. Existe otro método para cambiar el contexto de una función y es el método “apply” que recibe como primer parámetro el objeto que utilizará como contexto al ejecutarse y como segundo parámetro un arreglo con los parámetros que se necesiten pasar. El siguiente ejemplo muestra como usar esta función:

```

1. function notGlobal(){
2.     console.debug(this);
3. }
4.
5. notGlobal.apply(obj,[1,2,'3',true]);

```

Conclusiones

Este tema es muy importante pues es necesario comprender el uso de la variable “this” para los siguientes temas de este curso. El contexto puede ser una herramienta muy poderosa si se sabe utilizar correctamente, pero también puede darnos muchos dolores de cabeza si no se ha comprendido su uso.

Creación de objetos y métodos públicos

En JavaScript todo es un objeto, las funciones, Strings, arreglos, etc. Sin embargo muchos desarrolladores programan en este lenguaje sin utilizar los conceptos de la programación orientada a objetos. En JavaScript existen dos maneras de crear objetos, una es mediante el objeto “Object” y la otra es utilizando las llaves “{ }” y escribiendo la propiedad y su valor correspondiente. Veamos un ejemplo de esto:

```

1. //Se crea un objeto
2. var obj = new Object();
3.
4. //se crea una
5. obj.color = 0xffffffff; //Número hexadecimal ~>16777215
6. obj.click = function(){
7.     //Podemos acceder a la propiedad color
8.     console.debug(this.color);
9. }
10. //se llama al método click

```

```

11. obj.click();
12.
13. //Se crea un objeto
14. var obj = {
15.   //Se crea una propiedad color
16.   color: 0xfffff,
17.   //Este método pertenece al objeto obj
18.   click: function(){
19.     //Por lo tanto podemos acceder a la propiedad color
20.     console.debug(this.color);
21.   }
22. }
23. //Ejecutamos el método click
24. obj.click();

```

En el ejemplo anterior se muestra como es posible crear objetos de dos maneras diferentes dandonos la libertad de elegir la que más nos convenga. Es importante mencionar que **no podemos crear** nuevos objetos a través de los objetos anteriores.

Creación de objetos

En JavaScript no existe el concepto de “clase” por lo tanto creamos y heredamos objetos de otros objetos, este concepto lleva por nombre “Herencia de prototipo”. En JavaScript podemos instanciar una función que es utilizada como el constructor del objeto, recordemos que las funciones son objetos también, veamos el siguiente ejemplo:

```

1. //Una función Person (Objeto)
2. function Person(name,lastName){
3.   //Guarda el nombre y apellido en el contexto
4.   this.name = name;
5.   this.lastName = lastName;
6. }
7.
8. //Se crea una instancia de la función (Objeto) Person
9. var john = new Person("John","Doe");
10. //Se imprime el nombre y apellido
11. console.debug(john.name+" "+john.lastName);
12. //es una instancia del objeto Person
13. console.debug(john.constructor == Person);

```

En la última línea del ejemplo anterior se muestra la propiedad “constructor”, esta propiedad está presente en todos los objetos y siempre apunta a la función con la cual el objeto fue creado.

Métodos públicos

Los métodos públicos son completamente accesibles para las personas que utilizarán nuestro código y permite la comunicación del objeto con el exterior compartiendo o recibiendo información. Para realizar esto es necesario comprender la propiedad “prototype”, la cual es un objeto que actúa como referencia base para todas las copias que se realizarán de su padre, en otras palabras todas las propiedades y métodos que contenga la propiedad “prototype” estarán disponibles en cada instancia del objeto definido. La propiedad “prototype” es un objeto por lo tanto para agregarle propiedades lo hacemos de la misma forma que cualquier otro objeto, las propiedades agregadas serán totalmente públicas y accesibles para todos. Veamos el siguiente ejemplo:

```

1. //Objeto Animal
2. var Animal = function(name){
3.   this.name = name;
4. }
5. //Aregar un método público al objeto Animal
6. Animal.prototype.getName = function(){
7.   //El contexto es de la instancia creada
8.   return this.name;

```

```

9. }
10. //Método público
11. Animal.prototype.setName = function(name){
12.   //cambiar el valor de la propiedad name
13.   this.name = name;
14. }
15.
16. //Creamos una instancia de perro y gato
17. var dog = new Animal("Lucky");
18. var cat = new Animal("Milo");
19.
20. //Llamamos al método getName
21. console.debug(dog.getName());

```

Conclusiones

Crear instancias, propiedades y métodos públicos es un tema conocido para muchos desarrolladores, quizás para otros no, en el siguiente tema veremos como ocultar propiedades y métodos haciéndolos privados.

Métodos privados y métodos privilegiados

Este es un tema desconocido por muchos desarrolladores, algunos creen que esto no es posible hacerlo con JavaScript. En el tema de hoy veremos como ocultar propiedades y métodos en un objeto.

Propiedades y métodos privados

Para crear propiedades o métodos privados lo hacemos en el constructor. Los parámetros y variables ordinarias (las que se crean utilizando la palabra reservada “var”) así como los parámetros son propiedades privadas las cuales no pueden ser utilizadas desde el exterior del objeto. Los métodos privados son funciones dentro del constructor, este tipo de funciones no serán expuestas fuera del objeto. Veamos el siguiente ejemplo:

```

1. //The object User
2. var User = function(options){
3.   //private properties
4.   var name = options.name;
5.   var password = options.password;
6.   //public property
7.   this.user = options.user;
8.
9.   //private method
10.  function privateMethod(){
11.    console.debug('im a private method!')
12.  }
13. }
14.
15. //new instance of User
16. var u = new User({name:'Crysfel',password:'****',user:'cvilla'});
17.
18. console.debug(u.user); //cvilla
19. console.debug(u.name); //undefined
20. console.debug(u.privateMethod()); //throws error

```

En el ejemplo anterior se han agregado dos propiedades y un método privado, cuando intentamos imprimirlas en la consola de Firebug podemos ver que no se imprimen y el método lanza un error; esta es la manera en como se oculta la información dentro de un objeto.

Métodos privilegiados

Douglas Crockford le llama métodos privilegiados a los métodos que son públicos y que a la vez pueden acceder a las propiedades privadas y manipularlas, estos métodos son creados dinámicamente pues son agregados al objeto en tiempo de ejecución. Veamos un ejemplo:

```

1. var Person = function(options){
2.   //private properties
3.   var name = options.name
4.   var birthYear = options.birthYear;
5.   //private method
6.   var calculateAge = function(){
7.     var today = new Date();
8.     return today.getFullYear() - birthYear;
9.   }
10.  //Privileged method
11.  this.getAge = function(){
12.    return calculateAge(); //calling private method
13.  }
14. }
15. //new Person instance
16. var p = new Person({name:'Peter', birthYear:1983});
17.
18. console.debug(p.getAge());// the age
19. console.debug(p.name);// undefined
20. console.debug(p.birthYear);// undefined

```

Los métodos privilegiados son una excelente herramienta y muy importantes para poder acceder a la información privada, de esta manera es como exponemos la información que sea necesaria. Es importante mencionar que los métodos privilegiados son creados en tiempo de ejecución pues están en el constructor del objeto y no en el prototipo (prototype object), en el siguiente ejemplo vamos a crear dinámicamente getter y setters para las propiedades que recibe el constructor mediante el parámetro "options".

```

1. var Animal = function(options){
2.   //a private method to do the job
3.   //and to create a new scope (see last chapter of this course)
4.   function createGetterAndSetters(properties,property){
5.     //attach the getter for the current property
6.     this['get'+property] = function(){
7.       return properties[property];
8.     }
9.     //attach the setter for the current property
10.    this['set'+property] = function(value){
11.      properties[property] = value;
12.    }
13.  }
14.  //iterate through the options
15.  for(var property in options){
16.    //call the private method with the right context
17.    createGetterAndSetters.call(this,options,property);
18.  }
19. }
20. //two different instances of Animal object
21. var zul = new Animal({breed:'Chihuahua',gender:'female',name:'Zul'});
22. var rocky = new Animal({breed:'Beagle',gender:'male',name:'Rocky'});
23.
24. console.debug(zul.getbreed());//Chihuahua
25. console.debug(zul.getname());//Zul
26. console.debug(zul.name);//undefined
27. console.debug(rocky.getbreed());//Beagle
28. console.debug(rocky.getgender());//male
29. console.debug(rocky.getname());//Rocky
30. console.debug(rocky.breed)//undefined

```

En el ejemplo anterior se crean dinámicamente los getters y setters de las propiedades que se reciben como parámetro en el constructor, haciendo uso de un método privado para realizar el trabajo además de crear un nuevo scope cuando se manda a ejecutar permitiendo de esta forma acceder a los valores correctos, (si tienes alguna duda de este concepto te recomiendo leer el tema donde se ha explicado con mayor detalle el concepto del scope y el problema que existe cuando creamos dinámicamente métodos). Otro punto importante de comentar es en el ciclo for, ya que estamos invocando al método privado asignándole el contexto adecuado, de esta manera nos aseguramos que la variable "this" dentro del método privado esté haciendo referencia al objeto "Animal" (si tienes dudas al respecto te recomiendo leer el tema sobre el contexto).

Conclusiones

Es posible crear métodos y propiedades privadas así como los métodos privilegiados gracias a los "closures" que maneja JavaScript, ya que es posible acceder a las variables externas de la función contenedora aún cuando ésta haya finalizado su ejecución. Los métodos privados y privilegiados solo pueden ser definidos en el constructor (por los conceptos mencionados anteriormente), los métodos públicos pueden ser agregados en cualquier momento utilizando el objeto "prototype". Hemos visto como ocultar la información en un objeto, además de que hemos utilizado varios conceptos de temas anteriores como el scope, el contexto, closures; es importante tener en cuenta estos conceptos pues los utilizaremos más adelante. También veremos algunas utilerías que Ext JS nos proporciona para manejar de manera más sencilla estos términos.

Paquetes y namespace

Es importante tener en cuenta que otros desarrolladores o usuarios pueden utilizar nuestro código, además de que necesitamos crear código que pueda ejecutarse con alguna otra librería sin tener ningún problema. En mi experiencia me ha tocado trabajar en proyectos donde las variables están declaradas en el "global scope" generando problemas realmente extraños y difíciles de rastrear, es en esos momentos cuando te das cuenta de la importancia de empaquetar nuestras aplicaciones.

JavaScript no tiene un palabra reservada que nos permita empaquetar nuestros objetos como en lenguajes Java o C#, pero tomando en cuenta que los objetos pueden alojar otros objetos podemos solucionar este problema de la siguiente manera.

```
1. //Se crea el objeto com
2. var com = {};
3. //se le agrega el objeto quizzpot
4. com.quizzpot = {};
5. //se crea el "paquete"
6. com.quizzpot.tutorial = {};
7. //Se agrega un constructor de objetos
8. com.quizzpot.tutorial.Person = function(options){
9.     this.name = options.name;
10. }
11. //se crea una instancia
12. var p = new com.quizzpot.tutorial.Person({name:'John'});
13. //se imprime en consola
14. console.debug(p);
```

De esta manera se crea un espacio robusto de nombres, con esto nos evitaremos que alguien más cree un objeto "Person" y sobrescriba el nuestro. Si cada vez que creamos una función constructora creamos todo el paquete podemos sobrescribir paquetes existentes, para evitar esto debemos verificar si el paquete ya existe, de ser así utilizar el existente y agregar los que sean necesarios, eso lo hacemos de la siguiente manera:

```
1. //verificamos si existe la variable "com" para usarla,
2. //si no existe creamos un objeto vacío
3. var com = com || {};
4. //hacemos el mismo procedimiento
5. com.quizzpot = com.quizzpot || {};
```

```

6. com.quizzpot.tutorial = com.quizzpot.tutorial || {};
7.
8. //Creamos el constructor del objeto
9. com.quizzpot.tutorial.User = function(options){
10.   this.nickname = options.nickname;
11. }
12.
13. //Se crea una instancia
14. var p = new com.quizzpot.tutorial.Person({name:'John'});
15. var u = new com.quizzpot.tutorial.User({nickname:'stock'});
16. //Se imprime en consola
17. console.debug(u);
18. console.debug(p);

```

Para evitar todo el proceso de comparar la existencia de los objetos a la hora de formar los paquetes, podemos utilizar una herramienta de Ext JS, de la siguiente manera:

1. Ext.namespace('com.quizzpot.tutorial');
2. //o bien el siguiente atajo
3. Ext.ns('com.quizzpot.tutorial');

De esta manera creamos el paquete deseado y si existe no lo sobrescribe, simplemente utiliza el anterior.

Conclusiones

Aunque en JavaScript no exista una palabra reservada para crear paquetes, como es el caso de otros lenguajes, si podemos implementar esta técnica ya que sí se puede alojar un objeto dentro de otro objeto. Es importante recordar que al definir el nombre de los “paquetes”, debemos verificar si existen para no sobreescribirlos.

Herencia en JavaScript

JavaScript tiene una forma única de crear objetos y realizar herencia entre ellos, a esta forma se le llama “prototypal inheritance”, básicamente un objeto puede heredar métodos y propiedades de otros objetos creando un prototipo para generar nuevos objetos. La herencia de prototipo se realiza utilizando la propiedad “prototype” que se encuentra en todos los objetos. En JavaScript la herencia es simple pero con un poco de trabajo se puede obtener herencia múltiple; en este tema se muestra como realizar la herencia simple. Es importante mencionar que la propiedad “prototype” sólo puede heredar de otros objetos y no de otros prototipos o funciones constructoras. A continuación se muestra un ejemplo para una mejor comprensión:

```

1. //Super "class"
2. var Animal = function(type){
3.   this.type = type;
4. }
5.
6. Animal.prototype.getType = function(){
7.   return this.type;
8. }
9.
10. var Dog = function(options){
11.   this.breed = options.breed;
12. }
13.
14. //Inheritance
15. Dog.prototype = new Animal('Dog');
16.
17. //attach methods to the Dog "class"
18. Dog.prototype.run = function(){
19.   console.debug('the '+this.breed+' '+this.type+' is running!');
}

```

```

20. }
21. //new instance
22. var beagle = new Dog({breed:'Beagle'});
23.
24. //calling a method of the super "class"
25. console.debug(beagle.getType());
26. beagle.run();

```

La parte más importante del ejemplo anterior es donde se realiza la herencia, “Dog.prototype = new Animal('Dog');”. La variable Dog hace referencia a la función constructora del objeto Dog, “new Animal()” está creando un objeto Animal el cual es asignado al prototipo de la función constructora del objeto “Dog”; de esta forma el objeto Dog contendrá todos los método y propiedades el objeto “Animal” cuando se creen nuevas instancias.

Conclusión

La herencia simple es algo que utilizaremos cuando desarrollemos componentes, es necesario que tengamos este concepto en mente.

Quizz: Conceptos fundamentales de JavaScript (Parte I)

El capítulo de programación orientada a objetos ha terminado, es hora de realizar un pequeño “Quizz”. Ya que el material es extenso he decidido realizar el quizz en dos partes para que no sea tan grande. Este es el quizz de la primera mitad del tema, próximamente se publicará la segunda mitad, suerte!

1.- ¿Que es una referencia?

- Es una variable común.
- Es un puntero al lugar exacto donde se encuentra un objeto.
- Es un objeto almacenado en la memoria.
- Es un espacio de memoria reservado por el compilador.

2.- ¿En JavaScript es posible que una referencia apunte a otra referencia?

- Si
- No
- No sé

3.- Dado el siguiente código, ¿qué se imprimirá en la consola de Firebug?

```

var array = ['Ext JS','Mootools','jQuery'];
var ref = array;

array.push('prototype');
ref.pop();
ref.pop();

console.debug(array.length);
console.debug(ref.length);

```

- Un 4 y un 2.
- Un 2 y un 2.
- Un 4 y un 3.
- Un 3 y un 2.

4.- ¿Qué es el scope?

- Es una palabra reservada para declarar variables locales.
- Es el espacio en memoria donde se encuentra un objeto.
- Es un objeto global.
- Es el alcance o el área donde una variable puede utilizarse.

5.- ¿En JavaScript el scope de una variable es de bloques?

- Si, la variable sólo existe en el bloque definido.
- Si, las variables pueden ser definidas localmente.
- No, las variables pertenecen a la función en cuestión.
- No, las variables siempre son globales.

6.- Dado el siguiente código, ¿qué se imprimirá en la consola de Firebug al ser ejecutado?

```
function test(){  
    if(true){  
        var x = 2;  
    }  
    var y = x*5;  
    console.debug(y);  
}  
test();
```

- Se genera un error por que la variable "x" no está definida fuera del bloque "if"
- NaN por que "x" no está definida y al multiplicarla con "5" el resultado no es un número válido.
- 0 porque "x" no está definida por lo tanto vale 0.
- 10 por que la variable "x" pertenece a la función y no al bloque "if".

7.- ¿Qué se imprime en la consola de Firebug al finalizar la ejecución del siguiente código?

```
function test(){  
    x = 2;  
}  
test();  
  
var y = x*5;  
console.debug(y);
```

- Se genera un error por que la variable "x" sólo pertenece a la función "test".
- NaN por que la variable "x" no ha sido definida.
- 10 por que la variable "x" fue definida en el global scope ya que no utiliza la palabra reservada "var".
- 5 por que la variable "y" omite el contenido de la variable "x".

8.- Es un problema que puede solucionar con el uso de closures:

- Ocultar variables globales ya que puede servir como un contenedor.
- Generar métodos o eventos dinámicamente.

- Funciones internas pueden acceder a variables que se encuentran en el scope de su padre o contenedor.
- Todas las anteriores

9.- ¿La variable “this” a quién hace referencia?

- Únicamente al contexto global.
- Es una referencia al objeto global “window”.
- Al objeto o contexto en el cuál se está ejecutando el código.
- A la función o método que está siendo ejecutada.

10.- Dado el siguiente código menciona cual es el resultado que se imprime en la consola de Firebug al terminar su ejecución.

```
var obj = {
    value: 4,
    init: function(){
        var result = this.value/2;
        console.debug(result);
    }
}
```

```
window.onload = obj.init;
```

- 2, ya que la variable “value” fue definida en el objeto con valor de 4.
- NaN, ya que la función “init” está ejecutándose en un contexto diferente.
- La función “init” nunca se ejecuta.
- Ninguna de las anteriores.

11.- ¿Cuáles son los métodos nativos de JavaScript para invocar una función y asignarle su contexto de ejecución?

- El método “call” y el método “apply”.
- El método “invoque” y el método “context”.
- El método “delegate” y el método “bind”.
- No existen esas funciones

12.- ¿Qué es lo que se imprime en la consola de Firebug cuando termina la ejecución del siguiente código?

```
var obj = {
    x: 4,
    y: 10
}

function test(){
    this.z = this.x + this.y;
}
test.call(obj);

console.debug(obj.z);
```

- undefined, ya que la propiedad “z” no se ha definido en el objeto “obj”.
- 14, por que al ejecutar la función “test” se definió la propiedad “z” al objeto “obj”.
- Al realizar la suma de “x” y “y” se genera un error pues estas variables no existen en ese contexto.
- NaN porque las variables que se están sumando no han sido definidas.

Quizz: JavaScript orientado a objetos (Parte II)

Esta es la segunda parte del quizz del tercer capítulo del curso, los temas examinados son: creación de objetos y métodos públicos, métodos privados y privilegiados, paquetes y namespace, herencia simple con JavaScript. Si estos conceptos te son desconocidos te sugiero estudiar antes de presentar esta prueba, suerte!

1.- *En JavaScript las funciones son objetos.*

- Ciento, las funciones son un objeto pues tienen propiedades y métodos.
- Falso, las funciones son rutinas o métodos pero no son objetos.

2.- *¿Cuál es la forma correcta de crear un objeto?*

- Únicamente utilizando la palabra reservada “Object”.
- Podemos utilizar las llaves “{propiedad:value}” o la palabra reservada “Object”.
- JavaScript no es un lenguaje orientado a objetos por lo tanto no se pueden crear objetos.

3.- *¿Cómo podemos crear instancia de un objeto?*

- Mediante la palabra reservada “class”.
- Usando un “instance of” para crear la instancia requerida.
- Aplicando un “new” a la función constructora del objeto.
- Es imposible crear objetos en JavaScript.

4.- *¿Cómo definimos un método público?*

- Anteponiendo al método la palabra reservada “public”.
- Utilizando la palabra reservada “visible”.
- Utilizando la propiedad “prototype” de la función constructora y agregándole el método deseado.
- Ninguna de las anteriores.

5.- *¿Es posible crear métodos y propiedades privadas?*

- Por supuesto que si.
- Claro que no.

6.- *Dado el siguiente código, cual es el resultado al ser ejecutado?*

view plaincopy to clipboardprint?

```

1. var User = function(options){
2.     this.user = options.user;
3.     function sayHi(){
4.         console.debug('hi people!');
5.     }

```

- ```
6. }
7. var u = new User({name:'Crysfel'});
8.
9. u.sayHi();
 En la consola de Firebug se imprime "hi people!".
 Se genera un error cuando intenta ejecutar el método sayHi porque es un método privado.
```

**7.- De acuerdo con “Douglas Crockford”, ¿qué es un método privilegiado?**

- Es lo mismo que los métodos públicos.
- Son aquellos métodos que pueden realizar llamadas Ajax al servidor.
- Son métodos públicos que pueden acceder a las variables privadas.
- No tengo idea quien es esa persona ni de sus aportaciones.

**8.- ¿En qué momento se crean los métodos privilegiados?**

- Estos métodos se crean aleatoriamente en todos los objetos en JavaScript.
- Son métodos creados dinámicamente en tiempo de ejecución pues se encuentran dentro de la función constructora.
- Son creados de la misma forma que los métodos públicos.
- Ninguna de las anteriores.

**9.- ¿Cómo podemos crear un “paquete”?**

- Utilizando la palabra reservada “package” al inicio de nuestro código.
- Es imposible crear un “paquete” con JavaScript.
- Creando objetos dentro de otros objetos.
- Escribiendo el código dentro de un objeto llamado “namespace”.

**10.- ¿Cuales son los problemas que se presentan por no utilizar “paquetes”?**

- Podemos borrar o sobre escribir fácilmente el trabajo de nuestros compañeros de equipo.
- La compatibilidad entre librerías de terceros.
- Podemos sobre escribir variables y obtener resultados no deseados y difíciles de depurar.
- Todas las anteriores

**11.- ¿Cuál es la forma de crear “paquetes” con Ext JS?**

- Ext.namespace('paquete.que.necesitemos') y Ext.ns('paquete.que.necesitemos')
- Ext.package('paquete.que.necesitemos')
- Namespace('paquete.que.necesitemos') y Package('paquete.que.necesitemos')

# Conceptos de ExtJS

## ¿Qué es el “xtype”?

Cuando iniciamos nuestro aprendizaje con ExtJS vemos algunos conceptos que no entendemos, uno de estos es el xtype, existe una confusión sobre lo que en realidad es esta propiedad, para que sirve y en que escenarios conviene utilizarla. En este tutorial veremos el uso adecuado de la propiedad “xtype”, trataremos de explicar de la mejor manera las diferencias, las ventajas y sobre todo el uso correcto de esta configuración.

### Definición

En simples palabras, el xtype es un nombre simbólico que se le asigna a un componente o bien es un atajo o nombre corto asignado a cada componente para crearlo de manera “lazy”.

```
1. {
2. xtype: "panel",
3. width: 300,
4. height: 250,
5. html: "I'm a lazy Panel"
6. }
```

### ¿Cuál es la manera “lazy”?

Antes de que apareciera la versión 2.0, para crear un componente necesitabas crear una instancia de éste utilizando el operador “new”, luego podías renderizarlo o asignárselo a otro componente, esto nos ocasionaba un problema que describiré a continuación.

Imagina que necesitas tener veinte “Tabs” en una pantalla, antes de la versión 2.0 tenías que crear veinte instancias del componente Panel y luego asignárselas al “TabPanel” para que este las “renderice”, el problema aquí es que un solo “Tab” se visualiza a la vez y crear veinte instancias es un desperdicio de memoria ya que posiblemente el usuario de tu aplicación solamente utilizará una o dos pestañas, además de que el rendimiento de tu aplicación se verá afectado.

¿Por qué no ir creando cada “Tab” cuando se necesite?

Para solucionar este problema en la versión de ExtJS 2.0 aparece el concepto de “xtype”, permitiéndonos crear los componentes de una manera “lazy”, en otras palabras permitiéndonos ir creando los componentes justo cuando sean desplegados en pantalla.

En realidad ExtJS no nos permite inicializar nuestros componentes de manera “lazy” por defecto al utilizar el “xtype”, de acuerdo con la documentación y con la WIKI oficial si es posible, pero la realidad es otra, gracias a nuestro amigo Florian Cargoet por señalarnos este error.

### ¿Cómo puedo mejorar el rendimiento de mi aplicación?

Ya que utilizando “xtype” sucede exactamente lo mismo que si crearamos instancias de los componentes una por una, ¿cómo podríamos mejorar el rendimiento de nuestra aplicación? una solución es ir cargando los componentes via Ajax, aquí es donde es realmente útil el uso de “xtype”, para mayor información sobre esta técnica te recomiendo leer el siguiente post, más adelante hablaré a detalle sobre esto.

### Ventajas de utilizar la configuración xtype

En el ejemplo anterior se ha mostrado la principal ventaja de utilizar el “xtype”, y es el hecho de ir creando nuestros objetos justo cuando lo necesitamos, esto nos permite mejorar el rendimiento de nuestras aplicaciones, mejorará considerablemente el tiempo en que se cargan nuestras pantallas, especialmente si estamos utilizando muchos componentes. Otra ventaja es que escribiremos menos código, eso significa que al comprimir nuestros archivos JS resultarán un poco más ligeros, quizás no es mucha la diferencia pero si hay que tomarlo en cuenta.

## **Creación de nuestro propio “ xtype”**

ExtJS cuenta con varios componentes los cuales tienen su propio “ xtype” definido, pero muchas veces nosotros decidimos crear una extensión o componente el cual deberíamos asignarle su propio “ xtype”, esto es posible y es muy sencillo de realizar ya que únicamente necesitas hacer lo siguiente:

1. //Registrar xtype
2. Ext.reg(" xtype ", Application.MyComponente);

De esta manera podrías crear instancias de tu componente de manera “lazy”.

## **Conclusiones**

Una vez que tenemos claro el uso del “ xtype” es recomendable utilizarlo siempre que nos sea posible, ya hemos visto las ventajas de utilizarlo y creo que nos beneficiará considerablemente en el desarrollo de nuestras aplicaciones, podemos encontrar los “ xtype” disponibles en la documentación.

## **Trabajando con el DOM**

Se muestran las utilerías de Ext para manipular el DOM, trabajar con efectos y animaciones, realizar búsquedas en el DOM y aplicar eventos.

## **Conociendo al objeto “Element”**

Cuando necesitamos trabajar con elementos del DOM (Document Object Model) es de suma importancia conocer el objeto “Element” pues contiene métodos que nos facilitará el trabajo.

### **Esperar cuando el DOM esté listo para usarse**

Hemos visto en los primeros temas de este curso como lograr esto, en esta ocasión utilizaré un objeto debidamente empaquetado para explicar el funcionamiento del objeto “Element”, de esta manera evitamos conflictos con cualquier otro código de terceros.

1. //se crea el “paquete”
2. Ext.namespace('com.quizzpot.tutorial');
3. com.quizzpot.tutorial.Element = {
4.     init: function(){
5.         //el código del tutorial irá aquí
6.     }
7. }
8. //Cuando el DOM esté listo se invoca la función “init”
9. Ext.onReady(com.quizzpot.tutorial.Element.init,com.quizzpot.tutorial.Element);

Es importante mencionar que el segundo parámetro que se le pasa al evento “onReady” es el contexto donde se ejecutará.

### **Obtener un elemento**

Normalmente utilizamos el método “document.getElementById” para tomar elementos del documento mediante su identificador, con Ext JS podemos obtener los elementos de la siguiente manera:

1. var el = Ext.Element.get('my\_id');
2. // o simplemente el atajo
3. var el = Ext.get('my\_id');

Existen numerosas ventajas al utilizar este método ya que no solamente regresa el elemento del DOM sino que retorna un objeto con métodos y propiedades que podemos utilizar para manipularlo, por ejemplo para agregar o quitar clases CSS lo hacemos de la siguiente manera:

1. //tomamos el elemento “panel”
2. var el = Ext.get('panel');
3. //se le agrega la clase “element”
4. el.addClass('element');
5. //le quitamos la clase “anotherClass” al elemento
6. el.removeClass('anotherClass');

También podemos agregarle estilos al elemento de una manera muy sencilla utilizando el método “setStyle” de la siguiente manera:

```
1. //un estilo a la vez
2. el.setStyle('background-color','#CFE5FA');
3. el.setStyle('border','1px solid #99BBE8');
4. // o varios al mismo tiempo
5. el.setStyle({
6. 'background-color','#CFE5FA',
7. 'border','1px solid #99BBE8'
8. });
```

Existen muchos métodos más que están disponibles y que nos pueden ayudar para trabajar eficientemente. En el siguiente ejemplo se muestran varios de ellos:

```
1. //Centrar el elemento en la pantalla
2. el.center();
3. //cambiar la opacidad del elemento a 85%
4. el.setOpacity(.85);
5.
6. //obteniendo el padre del elemento
7. var parent = el.parent();
8. //agregando algunos estilos al padre
9. parent.setStyle({
10. 'background-color':'#ccc',
11. 'font-family':'Trebuchet MS',Arial,sans-serif',
12. 'font-size':'.9em'
13. });
14.
15. //el contenido se hace no seleccionable
16. el.unselectable();
17.
18. //elimina el elemento del DOM
19. el.remove();
```

### Mejorado el rendimiento

Si solamente necesitamos modificar alguna propiedad de un elemento en una sola línea de código y no necesitamos tener una referencia a ese objeto es conveniente utilizar el método “fly” de la clase “Element”, pues nos permite ahorrar memoria del explorador ya que no crea una instancia de la clase Element para que el objeto sea manipulado sino que utiliza la misma memoria una y otra vez, por ejemplo para cambiar un estilo sería de la siguiente manera:

```
1. Ext.Element.fly('footer').setStyle('border','1px solid #aaa');
2. // o el atajo
3. Ext.fly('footer').setStyle('border','1px solid #aaa');
```

### Conclusiones

La clase “Ext.Element” puede ser utilizada con frecuencia en nuestros proyectos para manipular el DOM, por eso es importante conocer los métodos que contiene para utilizarlos cuando sea conveniente. Te recomiendo ver el API de esta clase para conocerla mejor.

## La clase “DomHelper”

Ext JS cuenta con una utilería para manejar el DOM de una manera sencilla, así que en este tema veremos como crear elementos, aplicar estilos e insertarlos en el DOM.

## **Material de apoyo**

Es necesario descargar el material de apoyo para realizar este tutorial, el cual es un documento HTML que incluye a la librería de Ext JS, algunos estilos y un “div” en el “body” para realizar los ejemplos de este tema, el documento en JavaScript está vacío.

## **Definiendo el namespace del tutorial**

Antes de comenzar a ver la clase DomHelper vamos a “empaquetar” el código que estamos por escribir para evitar problemas con otras librerías, esto lo hacemos de la siguiente manera:

```
1. Ext.namespace('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.DomHelper = {
4. init: function(){
5. //El código del tutorial irá aquí
6. }
7. }
8.
9. Ext.onReady(com.quizzpot.tutorial.DomHelper.init,com.quizzpot.tutorial.DomHelper);
```

Cuando el DOM esté listo para utilizarse se ejecuta la función “init”, por lo tanto vamos a escribir el código dentro de esta función para que sea ejecutado inmediatamente.

## **Crear elementos**

Es muy fácil crear elementos con esta utilería, simplemente tenemos que saber donde queremos insertar el nuevo elemento y definir el contenido que necesitemos insertar.

```
1. var list = Ext.DomHelper.append('content',{
2. id: 'my-list', tag:'ul', children:[
3. {tag:'li',children:[{tag:'a',href:'#',html:'Hello world!'}]},
4. {tag:'li',html:'Item 2'},
5. {tag:'li',html:'Item 3'},
6. {tag:'li',html:'Item 4'}
7.]
8. },true);
```

Mediante el método “append” insertamos un elemento al documento, el primer parámetro es el lugar donde se va a insertar, en este caso es el “ID” de un elemento (div) que está definido en el documento HTML pero puede ser un objeto de tipo “Ext.Element”; el segundo argumento es un objeto que va a ser insertado dentro del elemento asignado en el primer argumento, a este objeto se le ha definido un “id”, un “tag” y los “children”; la propiedad “id” actuará como el identificador del elemento, la propiedad “tag” se utiliza para definir el elemento a crear, en este caso una lista “ul” pero se puede definir cualquier elemento HTML que sea válido, opcionalmente se le pueden agregar hijos (children) mediante un arreglo de objetos.

## **Aplicar estilos**

Mediante la clase DomHelper podemos modificar los estilos de un elemento de una forma muy sencilla.

```
1. Ext.DomHelper.applyStyles('my-list',{
2. 'border':'5px solid #ddd',
3. 'padding':'5px',
4. 'background-color':'#f8f8f8'
5. });
6.
7. Ext.DomHelper.applyStyles('my-list','border:5px solid #ddd;padding:5px;background-color:#f8f8f8');
```

Únicamente es necesario especificar el elemento que deseamos modificar y como segundo parámetro definimos un objeto o un String con los estilos que necesitamos aplicarle al elemento.

## **Insertar elementos**

La clase DomHelper cuenta con varios métodos para insertar elementos al DOM, podemos utilizar estos métodos para especificar el lugar exacto donde queremos hacerlo.

1. Ext.DomHelper.insertBefore('my-list',{tag:'p',html:'Hey esto es un parrafo.'});
2. Ext.DomHelper.insertAfter('my-list',{tag:'p',html:'Soy otro parrafo, insertado mediante javascript.'});

Utilizando el método “insertBefore” podemos insertar el nuevo elemento antes del elemento que le especificamos, en este caso “my-list”, también se puede utilizar el método “insertAfter” para insertar el elemento después del elemento indicado.

## **Conclusiones**

Manipular el DOM es esencial para cuando creamos interfaces personalizadas, es por esto que es importante comprender y conocer el uso de esta clase que nos facilitará muchas cosas.

## **Buscar elementos del DOM con Ext JS**

En este tema vamos a construir unos “tabs”, para hacer esto es necesario buscar los elementos en el DOM para aplicarle la funcionalidad y estilos requeridos.

### **Material de apoyo**

El material de apoyo para este tema cuenta con un documento HTML, el JavaScript y una hoja de estilos para maquetar el contenido del HTML, lo que vamos a hacer es buscar elementos para aplicarles estilos, algunos eventos y como resultado construiremos unos tabs.

### **Ext.DomQuery**

Ext JS cuenta con el objeto DomQuery para buscar elementos en el DOM de una manera muy sencilla y rápida, además de que provee varios métodos que regresan una colección de nodos que satisfacen algún criterio de búsqueda o solamente el elemento que necesitamos que sea seleccionado por su identificador, además podemos asignarle el nodo sobre el cual debe realizarse la búsqueda.

### **Paso 1: Empaquetando el componente**

Lo primero que haremos es crear el objeto “Tabs” con su respectivo “paquete” para evitar sobrescribir algún otro componente.

1. Ext.namespace('com.quizzpot.tutorial');
- 2.
3. com.quizzpot.tutorial.Tabs = {
4.     init: function(){
5.         //Aquí irá el código del tutorial
6.     }
7. }
- 8.
9. Ext.onReady(com.quizzpot.tutorial.Tabs.init,com.quizzpot.tutorial.Tabs);

El componente anterior tiene una función “init” la cual se invoca tan pronto como el DOM esté listo para ser utilizado.

### **Paso 2: Dando estilo a los tabs**

Actualmente los tabs no tienen ningún estilo, son simples links a los cuales tenemos que agregarles una clase de CSS que definimos previamente en la hoja de estilos “domquery.css”. Necesitamos buscar todos los links que estén dentro de una lista y pertenezcan al menú, así que utilizamos el método “select” del objeto DomQuery, el cual regresa un arreglo con los elementos encontrados.

1. var items = Ext.DomQuery.select('div[id=menu] > ul li a');

Con la sentencia anterior lo que hicimos fue buscar los links “a” que estén dentro de un “div” cuyo “id” sea igual a “menu” y contenga inmediatamente una lista de tipo “ul” con un “li”. Ahora vamos a iterar el arreglo haciendo algunas modificaciones:

```

1. Ext.each(items,function(item,i){
2. item.id = item.innerHTML;
3. Ext.fly(item).addClass('menu-item');
4. Ext.fly('content'+item.id).setStyle('display','none');
5. });

```

Lo primero que se hace es utilizar la función “Ext.each” que recibe como primer parámetro el arreglo que se va a iterar y como segundo parámetro la función que se ejecutará por cada elemento en el arreglo, dentro de la función se le asigna un “id” al menú para que más adelante podamos distinguir los tabs, inmediatamente después se le agrega una clase “css” al elemento en cuestión (si no sabes que hace “Ext.fly” te recomiendo leer el tema anterior), por último se oculta el contenido que desplegará ese tab.

### Paso 3: Desplegando el tab inicial

Para desplegar el tab inicial debemos mostrar el contenido y seleccionar el tab correspondiente.

```

1. Ext.fly('contentHome').setStyle('display','block');
2.
3. var first = Ext.DomQuery.selectNode('#menu ul li a:last');
4. Ext.fly(first).addClass('item-selected');

```

Lo único interesante de esta porción de código es la parte donde buscamos el tab que seleccionaremos en este caso utilizamos la función “selectNode” del objeto DomQuery para buscar el último elemento de la lista “ul” contenidos en el div con id “menu” (observa que he utilizado “#menu” en lugar de “div[id=menu]”).

### Paso 4: Asignar estilos al div contenedor

Utilizando la función “Element.setStyle” vamos a asignarle la altura al div contenedor y aplicar un borde.

```

1. Ext.fly('content').setStyle({
2. 'height':'300px',
3. 'border':'5px solid #000',
4. 'border-top':'none'
5. });

```

Ya hemos hablado sobre el objeto Element en este curso, si tienes dudas sobre el código anterior te recomiendo leer el tema correspondiente.

### Paso 5: Crear el evento onClick

Vamos a iterar el arreglo “items” que creamos anteriormente para asignarle el evento clic a cada tab.

```

1. Ext.each(items,function(item){
2. this.onClick(item);
3. },this);

```

Lo más interesante de esta parte es el tercer argumento que recibe la función “Ext.each”, el cual es el contexto donde se ejecutará la función dada en el segundo parámetro, es muy importante asignar el contexto adecuado para poder invocar el método “this.onClick” que definiremos a continuación.

### Paso 6: Definiendo el Clic

Es necesario crear el método “onClick” dentro del objeto “Tabs”, el cual mostrará el contenido correcto cuando sea seleccionado un tab.

```

1. Ext.namespace('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.Tabs = {
4. init: function(){
5. var items = Ext.DomQuery.select('div[id=menu] > ul li a');
6.
7. // ... código removido para mejor compresión

```

```

8.
9. Ext.each(items,function(item){
10. this.onClick(item); //invocando el método onClick
11. },this); //Se asigna el contexto adecuado
12. },
13.
14. onClick: function(el){
15. // aquí irá el código para mostrar tabs
16. }
17. }

```

El código anterior muestra el lugar donde se define el método clic, el cual recibe un elemento del DOM. Lo primero que necesitamos hacer es crear el evento de la siguiente manera.

```

1. el.onclick = function(){
2. return false;
3. }

```

Con esta instrucción cada vez que se le dé clic a un tab esa función se ejecutará, ahí es donde pondremos las acciones que necesitemos realizar; la función regresa “false” para que no siga con el “href” del link.

### **Paso 7: Seleccionando el tab correcto**

Primero necesitamos remover la clase “item-selected” del tab que actualmente está seleccionado y asignarle esa misma clase al elemento al que le ha dado clic.

```

1. var old = Ext.DomQuery.selectNode('#menu ul li a[class*=item-selected]');
2. Ext.fly(old).removeClass('item-selected');
3. Ext.fly(this).addClass('item-selected');

```

### **Paso 8: Mostrar el contenido seleccionado**

Ahora que sabemos cual tab se le dio clic podemos mostrar el contenido correcto, pero antes debemos ocultar el contenido que actualmente se está mostrando.

```

1. var content = Ext.DomQuery.selectNode('#content > div{display=block}');
2. Ext.fly(content).setStyle('display','none');
3. Ext.fly('content'+this.id).setStyle('display','block');

```

La parte más importante está en la primera línea donde hacemos una búsqueda dentro del “div” que tiene por identificador “content”, el contenido que tenga el estilo “display=block”, es decir el contenido que se está visualizando actualmente, una vez encontrado se oculta y se muestra el nuevo contenido.

### **Conclusiones**

En este tema vimos como se puede utilizar la clase DomQuery para buscar elementos dentro del DOM, usando propiedades de CSS, clases de CSS, etiquetas de los elementos y atributos de algún elemento. Hemos creado además unos tabs sencillos pero funcionales, te recomiendo hacer experimentos con esta clase que es muy útil para cuando estemos desarrollando nuestras aplicaciones. Es importante mencionar que todo el ejercicio mostrado lo pudimos haber hecho con jQuery si lo utilizáramos como adapter del framework, queda a nuestra conveniencia utilizar lo que mas nos guste o conozcamos.

## **Aplicar efectos visuales a Elementos**

El usuario es la parte más importante de nuestros sistemas y debemos garantizar una buena impresión con nuestras aplicaciones. Ext JS cuenta con diferentes efectos visuales que nos permitirán atraer la atención del usuario para informarle el estado del sistema, en este tema se muestran los efectos que podemos utilizar.

La clase Ext.Fx es la responsable de agregar esta funcionalidad al objeto Element (del cual hemos hablado anteriormente), utilizamos los métodos de la clase Fx mediante las instancias del objeto Element.

## **Empaquetando el tutorial**

Para iniciar con el tutorial es necesario empaquetar el código que escribiremos, ya sabemos cuales son las ventajas de hacer esto.

```
1. Ext.namespace('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.Fx = {
4. init: function(){
5. //Aquí el código del tutorial
6. }
7. }
8. //Ejecutar la función "init" cuando el DOM esté listo para ser usado
9. Ext.onReady(com.quizzpot.tutorial.Fx.init,com.quizzpot.tutorial.Fx);
```

## **Fading**

El primer efecto visual que veremos se llama “fade” y existen dos tipos, fadeIn y fadeOut, los cuales ocultan o aparecen suavemente un elemento. Lo que vamos a hacer es crear un botón por cada efecto visual y al presionar ese botón ejecutaremos el efecto visual sobre el elemento que se encuentra en el material de apoyo.

```
1. //tomamos el elemento
2. var el = Ext.get('element');
3. el.center(); //lo centramos en la pantalla
4. //cuando se dé clic sobre el botón con id fadeInFx...
5. Ext.get('fadeInFx').on('click',function(){
6. el.setOpacity(0); //le damos opacidad 0 al elemento
7. el.fadeIn(); // para que aparezca suavemente
8. });
9. //al dar clic sobre el botón fadeOutFx...
10. Ext.get('fadeOutFx').on('click',function(){
11. el.fadeOut(); //desaparece suavemente
12. });
```

Este efecto visual puede ser utilizado para eliminar o agregar registros a una tabla o panel y de esta manera llamar la atención del usuario y hacerle notar que algo está pasando.

## **Frame**

El efecto “frame” genera un contorno alrededor del elemento que va creciendo y desapareciendo a la vez.

```
1. Ext.get('frameFx').on('click',function(){
2. el.frame();
3. });
```

Este efecto visual puede ser utilizado para indicarle al usuario que cierta parte de la pantalla requiere su atención; podemos cambiar el color que tiene por defecto por algún otro color como rojo o amarillo los cuales indican advertencia o error, adicionalmente nos permite especificar el número de veces que se quiera repetir el efecto visual, en este ejemplo serán tres repeticiones.

```
1. Ext.get('frameFx').on('click',function(){
2. el.frame('ff0000',3);
3. });
```

## **Ghost**

Normalmente este efecto visual se utiliza para remover elementos de la pantalla, ya que desaparece el elemento suavemente y lo desliza en la dirección definida, por defecto es hacia abajo.

```
1. Ext.get('ghostFx').on('click',function(){
2. el.ghost();
3. });
```

## **Highlight**

Este efecto visual es muy utilizado para mostrar mensajes a los usuarios, de esta manera la atención del usuario es captada al mensaje que se desea mostrar, por defecto el elemento “brilla” en color amarillo, pero puede ser configurado para resplandecer en otro color.

```
1. Ext.get('highlightFx').on('click',function(){
2. el.highlight('00ff77'); // cambiamos el color por defecto
3. Ext.fly('msg').highlight(); //resplandece en color amarillo
4. });
```

## **Puff**

Este efecto visual desaparece un elemento haciendo que “explote” y se difumine lentamente.

```
1. Ext.get('puffFx').on('click',function(){
2. el.puff();
3. });
```

Es útil cuando se desea eliminar algún elemento por ejemplo en un carrito de compras, o en alguna galería donde se requiere eliminar una imagen.

## **Scale**

Con este efecto visual podemos redimensionar un elemento, haciendo una transición de sus medidas actuales a las nuevas.

```
1. Ext.get('scaleFx').on('click',function(){
2. el.scale(50,50);
3. });
4. Ext.get('scale2Fx').on('click',function(){
5. el.scale(100,100);
6. });
```

El primer parámetro que recibe el método es el “width” y el segundo parámetro es el “height” del elemento.

## **Slide**

Existen dos tipos del efecto “slide”, el “slideIn” y el “slideOut”, estos efectos permiten aparecer y desaparecer elementos de la pantalla de una manera llamativa, puede ser una alternativa para alguno de los efectos anteriores.

```
1. Ext.get('slideInFx').on('click',function(){
2. el.slideIn(); //aparece el elemento
3. });
4. Ext.get('slideOutFx').on('click',function(){
5. el.slideOut(); //desaparece
6. });
```

Este efecto visual puede ser utilizado para crear algún menú vistoso, galerías, transición de imágenes, el límite es tu imaginación.

## **Shift**

Este efecto visual permite cambiar la posición, dimensiones y/o opacidad al mismo tiempo, es ideal para combinar varias propiedades que se deseen cambiar.

```
1. Ext.get('shiftFx').on('click',function(){
2. el.shift({
3. x: 100,
4. y: 200,
5. width: 200,
6. height: 200
```

```
7. });
8. });
```

Es importante mencionar que forzosamente se tiene que especificar alguna propiedad a cambiar de lo contrario al ejecutar el método sin argumentos no se realizará ningún efecto visual ya que no se le definió el comportamiento requerido.

### Posiciones

La mayoría de los efectos visuales mencionados soportan las siguientes direcciones de movimiento en los efectos.

tl: Esquina superior izquierda (The top left corner)  
t: El centro superior (The center of the top edge)  
tr: La esquina superior derecha (The top right corner)  
l: El centro izquierdo (The center of the left edge)  
r: El centro derecho (The center of the right edge)  
bl: La esquina inferior izquierda (The bottom left corner)  
b: El centro inferior (The center of the bottom edge)  
br: La esquina inferior derecha (The bottom right corner)

### Easing

Para aplicar aceleración o desaceleración a los efectos podemos especificarle algún tipo de “easing” de la siguiente lista:

easeNone  
easeIn  
easeOut  
easeBoth  
easeInStrong  
easeOutStrong  
easeBothStrong  
elasticIn  
elasticOut  
elasticBoth  
backIn  
backOut  
backBoth  
bounceIn  
bounceOut  
bounceBoth

```
1. Ext.get('scale3Fx').on('click',function(){
2. el.scale(200,200,{
3. easing:'elasticOut',duration:2
4. });
5.});
```

Con esto logramos animaciones más llamativas, dándole al usuario una mejor experiencia al utilizar nuestros sistemas.

### Conclusiones

Los efectos son una parte importante para mejorar la experiencia del usuario, algunos desarrolladores pueden pensar que esto es una pérdida de tiempo, pero la realidad es que los pequeños detalles son los que marcan la diferencia entre el éxito y el fracaso.

## Quizz: Trabajando con el DOM

Es hora de evaluar los conocimientos adquiridos sobre el DOM y como manipularlo de una manera sencilla utilizando el Framework de Ext JS. ¡Suerte!

### 1.- ¿Cuál es la manera correcta para esperar a que el DOM este listo para usarse?

- Ext.addEvent(document,'ready',functionToExecute);
- Ext.on('ready',functionToExecute, scope );
- Ext.onReady(functionToExecute, scope );
- Ext.ready(functionToExecute, scope );

### 2.- ¿Cómo podemos obtener elementos del DOM?

- Ext.getElement('id');
- Ext.get('id');
- Ext.getElementById('id');
- Ext.getCmp('id');

### 3.- ¿Para que sirve el método Ext.fly?

- Es lo mismo que Ext.get, por lo tanto ni siquiera debería existir.
- Sirve para seleccionar una elemento del DOM utilizando selectores de CSS y XPATH.
- Con este método solamente tendremos el nodo del DOM y no un objeto Ext.Element.
- Para ahorrar memoria y hacer nuestra aplicación más eficiente ya que reutiliza la memoria al no crear nuevas instancias del objeto Ext.Element.

### 4.- El objetivo del componente Ext.DomHelper es:

- Ayudar al DOM para que se genera más rápido.
- Permite visualizar el HTML en forma gráfica.
- Nos ayuda para eliminar los nodos que ya no usamos.
- Nos facilita el manejo del DOM para insertar y crear elementos con JavaScript.

### 5.- Al ejecutar el siguiente código ¿que es lo que resulta en la variable "items"?

```
var items = Ext.DomQuery.select('div[id=menu] > ul li a');
```

- Todos los links que se encuentren dentro de un div que tenga como identificador "menu" y contenga un nodo de tipo "ul" el cual contenga "li".
- Los links de una lista con identificador "menu".
- Una lista de nodos de tipo "li".
- El selector es incorrecto.

**6.- Al ejecutar el siguiente código ¿que es lo que resulta en la variable “old”?**

```
var old = Ext.DomQuery.selectNode('#menu ul li a[class*=item-selected]');
```

- Un link donde se encuentre el Mouse.
- Un arreglo de links que se encuentren en el menú encapsulados en una lista de tipo “li”.
- El link que contenga una clase llamada “item-selected” y que esté dentro del elemento “menu” encapsulado en una lista de tipo “li”.
- No regresa nada porque la expresión es incorrecta.

## Mensajes y eventos

Se muestra como mostrar mensajes al usuario, como crear eventos sobre elementos del DOM

### Mensajes y alertas al usuario

Los mensajes son importantes para la interacción del usuario con nuestros sistemas, Ext JS cuenta con un componente especial para remplazar los típicos mensajes “alert” o confirmaciones por unos más llamativos.

#### Material de apoyo

Antes de seguir adelante es necesario descargar el material de apoyo para este tema, el cual contiene un HTML con varios botones y algunos estilos, un archivo JS vacío en el cual trabajaremos y una imagen la cual usaremos para el ejercicio, puedes además ver una demostración de lo que haremos en este tutorial.

#### Empaquetando el ejemplo del tutorial

Vamos a definir un “paquete” en donde estaremos trabajando, hemos visto las ventajas de realizar esto anteriormente.

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.Msg = {
4. init: function(){
5. //Aquí iré el código del tutorial
6. }
7. }
8.
9. //cuando el DOM este listo ejecutamos la función "init"
10. Ext.onReady(com.quizzpot.tutorial.Msg.init,com.quizzpot.tutorial.Msg);
```

#### Alertas

Mediante el componente “Ext.MessageBox” Ext JS nos proporciona los métodos necesarios para generar diferentes tipos de mensajes, uno de estos son los “alerts”, los cuales los generamos de la siguiente manera:

```
1. Ext.MessageBox.alert('Titulo','El mensaje que queremos dar');
2. //podemos usar el atajo
3. Ext.Msg.alert('Titulo','El mensaje que queremos dar');
```

Ahora vamos a agregarle el evento clic al botón “alert” que se encuentra en el HTML, dentro de este evento vamos a desplegar una mensaje de alerta.

```
1. Ext.get('alert').on('click',function(){
2. Ext.Msg.alert('Alerta','Esto es una alerta!');
3. },this);
```

De esta manera cuando presionemos el botón “Alert” se mostrará el mensaje que definimos en el código anterior.

#### Confirmación

Para mostrar una confirmación utilizamos el método “confirm”, en el siguiente ejemplo se le agrega el evento clic al botón de “confirmación” de la siguiente manera:

```
1. Ext.get('confirm').on('click',function(){
2. Ext.Msg.confirm('Confirmación','¿Estás seguro de querer hacer esto?');
3. },this);
```

Más adelante veremos como detectar el botón presionado y tomar las acciones correspondientes, por ahora nos conformaremos con mostrar el diálogo.

## Prompt

Este componente nos permite solicitar al usuario información mediante un diálogo que contiene una caja de texto. En el siguiente código le agregamos el evento clic al botón correspondiente y cuando se dispara el evento únicamente se despliega un mensaje solicitando el nombre del usuario.

```
1. Ext.get('prompt').on('click',function(){
2. Ext.Msg.prompt('Prompt','¿Cual es tu nombre?');
3. },this);
```

Más adelante veremos como capturar la información que el usuario ha introducido en la caja de texto.

## Wait

Mediante este método podemos mandar al usuario un mensaje de espera, este mensaje contiene una barra de progreso la cual avanza lentamente indicando que algo está sucediendo.

```
1. Ext.get('wait').on('click',function(){
2. Ext.Msg.wait('Cargando... porfavor espere!');
3. },this);
```

Si dejamos así el código anterior el mensaje se quedará ahí por siempre ya que no tiene opción de cerrarse, esto sucede por que nosotros debemos cerrarlo manualmente mediante una instrucción, normalmente lo cerraríamos cuando el proceso que lanzamos ha terminado, por ahora simplemente voy a mandar un “timeout” para cerrar el mensaje después de 6 segundos.

```
1. Ext.get('wait').on('click',function(){
2. Ext.Msg.wait('Cargando... porfavor espere!');
3. window.setTimeout(function(){
4. Ext.Msg.hide();
5. },6000);
6. },this);
```

Es importante mencionar que el método “hide” cierra cualquier mensaje que se esté mostrando en ese momento.

## Callbacks

Los métodos anteriores (excepto el wait) reciben como tercer parámetro una función, esta función será ejecutada cuando el usuario de clic en algún botón o cuando cierre el mensaje, esta función recibe como primer parámetro el botón al cual el usuario dio clic, y en el caso del método “prompt” recibe un segundo parámetro donde viene el texto que el usuario introdujo en la caja de texto, a continuación se muestra como podemos ejecutar ciertas instrucciones de acuerdo a lo que el usuario eligió en un mensaje de confirmación.

```
1. Ext.get('confirm').on('click',function(){
2. Ext.Msg.confirm('Confirmación','¿Estas seguro de querer hacer esto?',function(btn){
3. if(btn === 'yes'){
4. //si el usuario aceptó
5. alert('Has aceptado los terminos!')
6. }else{
7. //si el usuario canceló
8. alert('El usuario cancelo')
9. }
10. });
11. },this);
```

También podemos pasarle una función para ser ejecutada de la siguiente manera:

```
1. Ext.ns('com.quizpot.tutorial');
2.
3. com.quizpot.tutorial.Msg = {
4. init: function(){
5. // código removido para mayor compresión...
6. Ext.get('confirm').on('click',function(){
7. Ext.Msg.confirm('Confirmación','¿Estas seguro de querer hacer esto?',this.callback);
8. },this);
9.
10. // código removido para mayor compresión...
11.
12. },
13.
14. callback: function(txt){
15. alert(txt);
16. }
17. }
```

En el ejemplo anterior se creo una función dentro del objeto “Msg” la cual es invocada cuando el usuario da clic en algún botón del mensaje de confirmación.

### Mensajes personalizados

Si necesitamos desplegar algún mensaje personalizado, por ejemplo cambiar el ícono del mensaje o bien utilizar otros botones, entonces podemos hacerlo mediante el método “show” el cual recibe un objeto de configuración.

```
1. Ext.get('custom').on('click',function(){
2. Ext.Msg.show({
3. title: 'Personalizado', //<- el título del diálogo
4. msg: 'Esto es un mensaje personalizado!', //<- El mensaje
5. buttons: Ext.Msg.YESNO, //<- Botones de SI y NO
6. icon: Ext.Msg.ERROR, // <- un ícono de error
7. fn: this.callback //<- la función que se ejecuta cuando se da clic
8. });
9. },this);
```

Este es un mensaje que tiene un ícono de error, existen otros íconos que podemos utilizar:

```
1. Ext.Msg.ERROR //Ícono de error
2. Ext.Msg.INFO //Ícono de información
3. Ext.Msg.WARNING //Ícono de advertencia
4. Ext.Msg.QUESTION //Ícono de pregunta
```

Si necesitamos mostrar algún otro ícono debemos crear una clase CSS para poner la imagen como background de la siguiente manera:

```
1. .profile{
2. background:transparent url(profile.png) no-repeat;
3. }
```

Luego de esto se le especifica el nombre de la clase a la propiedad ‘icon’.

```
1. Ext.Msg.show({
2. title: 'Personalizado',
3. msg: 'Esto es un mensaje personalizado!',
4. buttons: Ext.Msg.YESNO,
5. icon: 'profile', // <- ícono personalizado
```

```
6. fn: this.callback
7.});
```

## Conclusiones

Ext JS cuenta con estos métodos para comunicarse con el usuario los cuales podemos usarlos en lugar de los típicos “alert” o “confirm” que vienen por defecto con el explorador, permitiendo hacer los mensajes más llamativos.

## Manejo de eventos sobre Elementos

Los eventos son muy importantes en nuestras aplicaciones, de esta manera podemos monitorear el comportamiento del usuario y el sistema reaccionará dependiendo de lo que esté sucediendo.

### El problema

El tema de los eventos se complica cuando necesitamos que nuestras aplicaciones funcionen correctamente en diferentes exploradores ya que existen diferencias en algunos exploradores en cuanto al manejo de eventos.

### La solución

Mediante Ext JS podemos evitarnos el trabajo de detectar el explorador y manejar los eventos de una u otra manera, simplemente es necesario agregar un “listener” en algún elemento o componente y Ext JS se encargará del resto.

```
1. //aquí le mostraremos al usuario un mensaje
2. var msg = Ext.get('msg');
3. //tomamos el elemento "frame"
4. var el = Ext.get('frame');
5.
6. //Agregamos el evento necesario
7. el.addListener('mouseover',function(event,element,options){
8. //ejecutamos todo lo que necesitemos cuando se dispare el evento
9. msg.dom.innerHTML = options.param1+'over!';
10. },this,{param1:'the mouse is '});
```

En el código anterior primero se obtiene el elemento “frame”, el cual es un div que se encuentra en el documento HTML, luego se le agrega un evento utilizando la función “addListener” la cual recibe como primer parámetro el evento que necesitamos monitorear (click, mouseover, mouseout, mousedown, etc...) y como segundo parámetro recibe una función, la cual es disparada cuando ocurra el evento especificado; opcionalmente podemos definir en el tercer parámetro el contexto donde se ejecutará la función del segundo parámetro, también podemos pasarle un objeto con las propiedades que necesitemos en el cuarto argumento.

Existe un atajo para la función “addListener”, podemos utilizar la función “on” ya que su funcionamiento es el mismo y de esta manera escribiremos menos.

```
1. //tomamos el elemento "frame"
2. var el = Ext.get('frame');
3.
4. //Agregamos el evento necesario
5. el.on('mouseover',function(event,element,options){
6. //ejecutamos todo lo que necesitemos cuando se dispare el evento
7. msg.dom.innerHTML = options.param1+'over!';
8. },this,{param1:'the mouse is '});
```

El código anterior hace exactamente lo mismo que el primer ejemplo, solo que en lugar de utilizar la función “addListener” se ha utilizado la función “on”.

A continuación voy a agregarle algunos eventos más al elemento para ver claramente el funcionamiento:

```

1. //cuando el mouse sale del elemento
2. el.on('mouseout',function(){
3. msg.dom.innerHTML = 'out!';
4. });
5.
6. //se dispara cuando se da click sobre el elemento
7. el.on('click',function(){
8. msg.dom.innerHTML = 'click!';
9. });
10.
11. //Si el botón del mouse es presionado sobre el
12. //elemento, éste dispara la función definida
13. el.on('mousedown',function(){
14. msg.dom.innerHTML = 'mouse down!';
15. });
16.
17. //Si se ha soltado el botón del mouse dentro del elemento
18. //se dispara esta función
19. el.on('mouseup',function(){
20. msg.dom.innerHTML = 'mouse up!';
21. });

```

### **Componentes**

Estas mismas funciones pueden ser utilizadas para monitorear eventos sobre los componentes (ventanas, grillas, árboles, menús, etc...) y su implementación es exactamente igual, simplemente hay que saber el evento que necesitamos monitorear (por ejemplo resize, load, collapse, etc.) dependiendo del componente.

A continuación muestro un ejemplo de cómo monitorear cuando una ventana es minimizada:

```

1. //suponemos que se ha creado una instancia
2. //del componente Ext.Window anteriormente
3. //y en este punto le asignamos un "listener"
4. //para cuando sea minimizada
5. win.on('minimize',function(win){
6. //aquí se realiza lo que se necesite
7. //cuando la ventana ha sido minimizada
8. });

```

### **Conclusiones**

Ext JS cuenta con una manera normalizada para poder ejecutar funciones cuando sucede determinado evento asegurándonos que funcionará sobre los diferentes exploradores.

## **Quizz: Mensajes y Eventos**

**1.- ¿Cuál es el componente que usamos para mostrar los mensajes?**

- Ext.Messages
- Ext.Alert
- Ext.Component.Message
- Ext.MessageBox

**2.- ¿Cuál es el alias o shortcut para el componente encargado de generar los mensajes?**

- Ext.info

- Ext.Msg
- Ext.show
- Ext.msg

**3.- ¿Cuál es el resultado al ejecutar el siguiente código?**

Ext.Msg.confirm('Confirmación','¿Estás seguro de querer hacer esto?');

- Se muestra un mensaje de alerta con un botón que cierra el diálogo.
- Se muestra un mensaje con dos botones para confirmar la pregunta realizada.
- Se muestra una mensaje de error para informar al usuario que algo anda mal.
- Ese código marcará un error ya que no existe el component Ext.Msg

**4.- ¿Qué es lo que sucede al ejecutar el siguiente código?**

Ext.Msg.wait('Cargando... por favor espere.');

- Bloquea la pantalla y muestra una barra de progreso con el mensaje indicado.
- Muestra un mensaje que puede ser cerrado en cualquier momento por el usuario.
- Se cambia el puntero del mouse con una animación que permite indicar al usuario que algo está sucediendo y necesita esperar.
- No existe el método "wait".

**5.- ¿Cuanto tiempo dura el mensaje “wait” (de la pregunta anterior)?**

- Un minuto y luego se cierra.
- Hasta que el servidor responda o termine el proceso que se está ejecutando.
- Nunca se cierra automáticamente, se tiene que cerrar manualmente.
- Espera a que el usuario presione el botón cerrar.

**6.- ¿Mediante que instrucción o método podemos quitar o cerrar los mensajes que se están mostrando?**

- Ext.hide()
- Ext.Msg.hide()
- Ext.Msg.close()
- Ext.msg.hide()

**7.- ¿Para qué sirven los callbacks?**

- Para ejecutar instrucciones cuando el mensaje sea cerrado por el usuario.
- Para iniciar el componente y asignar las propiedades necesarias.
- Esta propiedad no existe en los mensajes.

**8.- ¿Cómo podemos monitorizar los eventos ocurridos en los componentes?**

- Creando un hilo que este ejecutándose continuamente y reacciones a los cambios de estado.
- Agregando un “listener” al componente o elemento que necesitamos monitorizar.
- Poniendo un interceptor al componente o elemento.

- No podemos monitorizar ningún evento en los componentes.

#### 9.- ¿Cuál es el shortcut o atajo para el método “Ext.Element.addListener”?

- Ext.Element.addEvent
- Ext.Element.click
- Ext.Element.event
- Ext.Element.on

## Ajax y el objeto Store

Se muestra como realizar peticiones Ajax al servidor, se define el concepto del store y su importancia para el intercambio de información.

### El objeto Ajax, peticiones GET y POST

La comunicación con el servidor es parte vital en las aplicaciones Web. Enviar y recibir información sin que la página se actualice por completo es sencillo utilizando Ajax, en este tema veremos como realizar peticiones GET y POST utilizando el Framework de Ext JS.

### Material de apoyo

Para este tema es necesario descargar el material de apoyo, descomprimirlo y copiar los archivos al servidor Web que hemos instalado previamente, dentro de la carpeta curso creamos un directorio que se llame “ajax”, dentro de este pegamos los archivos del material de apoyo (ajax.html, ajax.js, ajax.php).

### Namespace

Lo primero que debemos hacer es “empaquetar” el código que escribiremos, asignarle un namespace donde será ejecutado, de esta manera evitamos coaliciones.

```
1. //el namespace para este tutorial
2. Ext.ns('com.quizzpot.tutorial');
3.
4. com.quizzpot.tutorialAjax = {
5. init: function(){
6. //el código va aquí
7. }
8. }
9.
10. //cuando esté listo el DOM ejecutamos la función "init"
11. Ext.onReady(com.quizzpot.tutorialAjax.init,com.quizzpot.tutorialAjax);
```

### Crear los eventos

El siguiente paso es crear un “listener” para el evento “click” de los botones que se encuentran en el documento HTML, y dentro de éste vamos a realizar las llamadas Ajax.

```
1. //cuando se de clic sobre el botón “json”...
2. Ext.fly('json').on('click',function(){
3. //hacemos una petición por Ajax al servidor
4. },this);
5. //hacemos lo mismo para los otros dos botones...
6. Ext.fly('xml').on('click',function(){
7. },this);
8. //que tenemos en el documento HTML
9. Ext.fly('html').on('click',function(){
10. },this);
```

## Ajax

Dentro de la función que se ejecutará cuando el usuario presione el botón “json” vamos a poner lo siguiente:

```
1. Ext.Ajax.request({ //dispara la petición
2. url: 'ajax.php', //la URL donde se realiza la petición
3. method:'GET', //El método HTTP usado para la petición
4. params:{format:'json'},//los parámetros que se usaran para la solicitud
5. success: this.log, //si se logró la comunicación, ejecuta la función "log"
6. failure: this.fail, //si falla, ejecuta la función "fail"
7. scope:this //especifica el contexto de las funciones anteriores
8. });

});
```

Como puedes ver en el código anterior, para realizar una petición por medio de Ajax, utilizamos el componente “Ext.Ajax”, no podemos crear instancias de este objeto ya que es un singleton, es por eso que lo podemos utilizar sin crear instancias, únicamente ejecutamos el método “request” y le pasamos un objeto con las configuraciones necesarias.

En la propiedad “url” se especifica donde será realizada la solicitud, en este caso “ajax.php” pero esto variará dependiendo de tus necesidades, algo muy importante por aclarar es que por ningún motivo deberías de ponerle parámetros a la url (ej: ajax.php?param=hola&id=2&module=/index), he visto muchas veces que algunos desarrolladores cometan esta “barbaridad”; esto no debe hacerse para pasar parámetros al servidor, ya que en el mismo objeto de configuración podemos definir los parámetros que necesitemos.

La propiedad “params” es responsable de enviar todos los parámetros que necesitemos, aquí es donde los definimos como un objeto ({name: ‘value’, param2:2}), usando un String (name=value&param2=2) o podemos asignarle una función la cual regresará los parámetros necesarios, esto es útil para cuando los parámetros son variantes y existen condiciones y reglas para enviar los parámetros correctos.

En la propiedad “method” se especifica el método http que utilizará la solicitud (POST, GET, PUT y DELETE), si no se especifica un tipo entonces toma GET si no se han enviado parámetros (usando la propiedad “params”), y POST si se han especificado parámetros.

Es importante mencionar que la propiedad “success” (del objeto de configuración) recibe una referencia a la función “log” la cuál aún no hemos escrito, esta función será ejecutada luego de que el servidor responda satisfactoriamente, esto quiere decir que puede demorar en ejecutarse dependiendo de la carga del servidor; igualmente la propiedad “failure” será ejecuta la función asignada cuando se produzca un error en el servidor o en la comunicación (404 not found, 403 forbidden, 500 server error, etc...).

Ahora vamos a realizar las peticiones para los otros botones que nos hacen falta:

```
1. //lo mismo para los otros dos botones...
2. Ext.fly('xml').on('click',function(){
3. Ext.Ajax.request({
4. url: 'ajax.php',
5. params:{format:'xml'}, //en formato XML
6. success: this.log,
7. failure: this.fail,
8. scope:this
9. });
10. },this);
11. //que tenemos en el documento HTML
12. Ext.fly('html').on('click',function(){
13. Ext.Ajax.request({
14. url: 'ajax.php',
15. success: this.log,
16. failure: this.fail
17. });
18. },this);
```

Si has notado, la configuración de estas peticiones no es exactamente igual, ha variado muy poco, esto con el propósito de mostrar la flexibilidad del componente.

### La función success

Es necesario escribir la función que se ejecutará cuando todo suceda correctamente, en este caso la función "log", la cual será un método del objeto "com.quizzpot.tutorial.Ajax".

```
1. , //<--no olvidar esta coma para separar entre los métodos
2.
3. ****
4. * If the request is successful ...
5. ****
6. log: function(response,options){//recibe la respuesta y el objeto de configuración
7. var el = Ext.get('response'); // tomar el LOG
8. var text = response.responseText; //remover el...
9. text = text.replace(/</g,'<'); // < y...
10. text = text.replace(/>/g,'>'); // >
11. el.select('p.newest').removeClass('newest'); // remover el ultimo update
12. Ext.DomHelper.append(el,'<p class="newest">'+text+'</p>'); //update el log
13. el.scrollTo('top',el.dom.scrollHeight); //posicionar el scroller al fondo
14. el.select('p.newest').highlight('00ff66',{duration:0.5}); //highlight el ultimo mensaje
15. }
```

Lo más importante del código anterior es que la función recibe dos parámetros, el primero es la respuesta del servidor el objeto XMLHttpRequest, y el segundo parámetro es el objeto de configuración, el resto del código no tiene mucha importancia comentarlo ya que debe ser remplazado por la funcionalidad que cada uno quiera implementar, pero si es importante mencionar que aquí es donde se debe tratar con la información regresada.

### La función failure

En caso de suceder un error debemos saber como manejarlo, el siguiente código es disparado cuando un error ha sucedido en la comunicación o bien en el servidor.

```
1. , //<-- coma separadora de los métodos
2.
3. ****
4. * If the request fails, log the error
5. ****
6. fail: function(response,options){
7. var el = Ext.get('response');
8. el.select('p.newest').removeClass('newest');
9. Ext.DomHelper.append(el,'<p class="newest">Error Status '+response.status+' '+response.statusText+': Opsss, there is something wrong! please try again</p>');
10. el.scrollTo('top',el.dom.scrollHeight);
11. el.select('p.newest').highlight('ff1111',{duration:0.5});
12. }
```

Simplemente se le informa al usuario que ha sucedido un error, nada complicado, pero cada quien es responsable de implementar lo que sea necesario en su proyecto.

### El servidor

Este curso es de Ext JS, por lo tanto no voy a explicar el código utilizado en el servidor a detalle puesto que es simplemente un ejemplo que no tiene utilidad alguna en el mundo real, pero si te diré que es lo que hace.

```
1. <?php
2. if(rand(1,4) == 1){
3. if(rand(1,2)==1)
4. header("HTTP/1.0 404 Not Found");
```

```

5. else
6. header("HTTP/1.0 403 Forbidden");
7. exit;
8. }
9.
10. $type = $_SERVER['REQUEST_METHOD'];
11. $msg = new Message("This is a ".$type." request!",true);
12.
13. $format = $type=='GET'? $_GET['format']:$_POST['format'];
14.
15. switch($format){
16. case 'xml':
17. header("Content-Type: text/xml");
18. echo $msg->toXML();
19. break;
20. case 'json':
21. header("Content-Type: text/plain");
22. echo $msg->toJSON();
23. break;
24. default:
25. header("Content-Type: text/html");
26. echo $msg->toHTML();
27. }
28.
29. class Message{
30. protected $msg,$success;
31.
32. public function __construct($msg,$success) {
33. $this->msg = $msg;
34. $this->success = $success;
35. }
36.
37. public function toXML(){
38. return "<response><success>".$this->success."</success><msg>".$this-
>msg."</msg></response>";
39. }
40.
41. public function toJSON(){
42. return "{success:".$this->success.",msg:\"".$this->msg."\"}";
43. }
44.
45. public function toHTML(){
46. return '<p>'.$this->msg.'</p>';
47. }
48. }
49. ?>

```

Primero aleatoriamente manda errores, ya sea un 404 o un 403, he creado una clase “Message” la cual toma un mensaje y una bandera en el constructor, cuenta con tres métodos los cuales retornan el mensaje en diferentes formatos, los cuales mediante un switch son invocados e impresos en el explorador.

### Conclusiones

El uso de Ajax es muy importante en nuestras aplicaciones, Ext JS ha revolucionado la manera en como desarrollamos nuestros sistemas o aplicaciones, este componente es fundamental en el Framework y es necesario conocerlo para usarlo apropiadamente, te recomiendo mires el API y juegues un poco con este componente.

## ¿Qué es un store y cómo funciona?

El tema de hoy es fundamental ya que el objeto Store es utilizado por los componentes que necesitan comunicarse con el servidor para mostrar la información, en este tema daré un vistazo rápido a lo más importante de este componente.

Un Store es un componente que almacena temporalmente información mediante registros, es utilizado como caché. Es importante mencionar que el Store contiene a otro componente capaz de leer e interpretar la información recibida, este lector es configurado antes de solicitar la información local o al servidor.

### Material de apoyo

Para este tema el material de apoyo es un HTML y un JS donde estaremos trabajando, así que es necesario descargarlo y copiarlos dentro de la carpeta “ajax” que creamos en el tema anterior la cual está dentro de la carpeta “curso” en el servidor Web que instalamos en el primer capítulo de este curso.

### Encapsulando el tutorial

Antes de comenzar con el ejemplo tenemos que encapsular el código que estaremos escribiendo para evitar coaliciones.

```
1. //El namespace para este tutorial
2. Ext.ns('com.quizzpot.tutorial');
3.
4. com.quizzpot.tutorial.Store = {
5. //Información dummy irá aquí
6.
7. init: function(){
8. //esto será ejecutado cuando el DOM esté listo
9. //crear el store aquí
10.
11. //cargar la información en el store aquí
12.
13. //crear los "listeners" de los botones aquí
14. }
15.
16. //crear el método "orderAsc" aquí
17.
18. // crear el método "orderDesc" aquí
19.
20. // crear el método "filter" aquí
21.
22. // crear el método "query" aquí
23.
24. // crear el método "count" aquí
25.
26. // crear el método "find" aquí
27.
28. // crear el método "log" aquí
29. }
30. //disparamos la función "init" cuando el DOM esté listo
31. Ext.onReady(com.quizzpot.tutorial.Store.init,com.quizzpot.tutorial.Store);
```

He comentado el lugar donde escribiremos el código del tutorial, con la intención de que tengas una idea global de la estructura final del código.

### La información

Para este ejemplo vamos a tomar la información de un arreglo, es importante mencionar que debemos crear un arreglo bidimensional el cual será “procesado” por el store que crearemos más adelante, este arreglo estará al inicio del objeto “com.quizzpot.tutorial.Store” de la siguiente manera:

```

1. data: [//información dummy para el ejemplo
2. [1,'Crysfel','Software developer','m',25],
3. [2,'Sasha','Figure skater','f',23],
4. [3,'Jack','Software Architect','m',35],
5. [4,'John','Javascript developer','m',24],
6. [5,'Sara','Tester','f',31]
7.],

```

La información está contenida en un arreglo el cual contiene otros arreglos con la información, cada arreglo interno será un registro donde la posición cero es el “identificador” del registro, la posición uno es el “nombre” de una persona, la posición dos la “ocupación”, la posición tres es el “género” de la persona y la posición número cinco es la “edad”.

### **Crear un Store con información local**

Ahora vamos a crear un “SimpleStore” con el que estaremos trabajando en este tutorial, esto lo hacemos de la siguiente manera:

```

1. //Creamos una instancia del SimpleStore
2. this.store = new Ext.data.SimpleStore({
3. fields: [//definimos los campos que tendrá...
4. {name:'name',mapping:1}, //cada registro...
5. {name:'occupation',mapping:2}, // y lo relacionamos...
6. {name:'gender',mapping:3}, // con una posición en el...
7. {name:'age',mapping:4}//arreglo que tiene la información
8.],
9. id: 0 //definimos la posición del ID de cada registro
10. });

```

Hasta este punto hemos creado el store, aún no tiene información pero ya es capaz de leer el arreglo que definimos anteriormente, la propiedad “fields”, que esta en la configuración del store, es donde se define el nombre de las propiedades de los registros mediante la propiedad “name” y se relaciona al arreglo con la información mediante la propiedad “mapping”, en este caso la propiedad mapping se le asigna la posición en el arreglo de donde sacará su contenido.

### **Cargar la información en el Store**

Introducir la información en el Store es muy fácil ya que estamos usando información local contenida en un arreglo. Para que el store pueda consumir el arreglo definido lo hacemos de la siguiente manera:

```

1. //cargar la información del arreglo
2. this.store.loadData(this.data);

```

Si todo ha salido bien ya podremos usar la información contenida en el store.

### **Crear los “listeners” de los botones**

Lo siguiente que haremos es crear los “listeners” del evento clic de cada botón que hay en el documento html.

```

1. Ext.fly('personBtn').on('click',this.find,this);
2. Ext.fly('txt').on('keyup',function(event,cmd){
3. if(event.getKey() === event.ENTER){ //cuando sea la tecla ENTER
4. this.find(); // realizamos la búsqueda
5. }
6. },this);
7. Ext.fly('ascBtn').on('click',this.orderAsc,this);
8. Ext.fly('descBtn').on('click',this.orderDesc,this);
9. Ext.fly('older2030Btn').on('click',this.query,this);
10. Ext.fly('older30Btn').on('click',this.filter,this);
11. Ext.fly('countBtn').on('click',this.count,this);

```

El código anterior ya es familiar para nosotros, de no ser así te recomiendo darle un repaso a los temas anteriores donde se habló al respecto, lo más importante a resaltar es que las funciones que se han asignado a cada evento no las hemos definido.

### Ordenar los registros

Ordenar la información es muy importante, y podemos hacerlo de una manera muy sencilla utilizando el método “sort”.

```
1. , //nota la coma separadora XD
2.
3. orderAsc: function(){
4. this.store.sort('name','ASC'); // ordenar en forma ascendente
5. this.store.each(function(record){//por cada registro...
6. this.log(record.get('name')); //imprime la propiedad "nombre"
7. },this);
8. this.log('_____');
9. }, // <--- esta coma es importante
10.
11. orderDesc: function(){
12. this.store.sort('name','DESC'); //Ordenar en forma descendente
13. this.store.each(function(record){ // por cada registro...
14. this.log(record.get('name')); //imprime la propiedad "nombre"
15. },this);
16. this.log('_____');
17. }
```

El método “sort” recibe como primer parámetro la propiedad por la que serán ordenados los registros y como segundo parámetro el tipo de orden, ascendente o descendente; una vez que se han ordenado se pueden recorrer los registros utilizando el método “each” del store, el cual iterá sobre los registros.

El método “log” no ha sido definido aún, lo haremos más adelante, por ahora puedes poner un “console.debug” para imprimir en la consola de Firebug.

### Filtrar registros en el store

En ocasiones es necesario filtrar la información contenida en el Store dependiendo algún criterio dado, en este ejemplo voy a realizar un filtro de las personas cuya edad sea mayor de 30 años; esto lo haré utilizando el método “filterBy”.

```
1. , // <--- La coma separadora
2.
3. filter: function(){
4. //filtrar a las personas...
5. this.store.filterBy(function(record,id){
6. return record.get('age') >= 30; //mayores a 30 años
7. });
8.
9. //por cada registro...
10. this.store.each(function(record){
11. //imprimir en el "log"
12. this.log(record.get('name')+' is older than 30 '+ (record.get('gender')=='f'? 'she':'he')+ ' is '+record.get('age'));
13. },this);
14. //limpiar los filtros
15. this.store.clearFilter();
16. this.log('_____');
17. }
```

El método “filterBy” acepta como primer parámetro una función que será ejecutada por cada registro del store, ahí es donde se ha definido la condición deseada (edad mayor a 30 años), cuando la función retorne “true” el registro será tomado en cuenta y cuando retorne “false” el registro será descartado.

Luego de aplicar el filtro al store se ejecuta la función “each”, es importante mencionar que la función “each” únicamente será ejecutada sobre los registros que han sido filtrados anteriormente ya que se le ha aplicado un filtro al store. Por último mediante la función “clearFilter” se limpian los filtros aplicados al store, permitiendo que todos los registros puedan ser utilizados nuevamente.

### Buscar registros

El método anterior nos proporciona un manera de buscar registros descartando los registros que no necesitamos, el método “queryBy” hace algo semejante pero la diferencia es que regresa los registros encontrados en una colección, esto nos puede ser más útil o quizás más claro que el método anterior.

```
1. //<--- La coma separadora
2.
3. query: function(){
4. //buscar gente mayor a 20 y menor que 30 años
5. var collection = this.store.queryBy(function(record,id){
6. return record.get('age') >20 && record.get('age')<30;
7. });
8.
9. //por cada item en la colección
10. collection.each(function(item,index){
11. //imprime su nombre y edad
12. this.log(item.get('name')+ ' is '+item.get('age')+ ' and '++(item.get('gender')=='f'?she':he')+ ' is y
13. ounger than 30');
14. },this);
15. this.log('_____');
```

Como puedes notar es muy semejante (por no decir igual) que el método anterior, la única diferencia es que regresa una colección con los registros que cumplen la condición especificada.

### Buscar por una propiedad

Si queremos buscar un registro único podemos utilizar el método “find” el cual recibe como primer parámetro la propiedad sobre la cual queremos realizar la búsqueda, como segundo parámetro recibe un “String” o una expresión regular con el criterio de búsqueda, el tercer parámetro es opcional y es el número de registro donde comenzará a realizar la búsqueda, en el cuarto parámetro que también es opcional definimos si la búsqueda será ejecutada en cualquier parte del texto y el quinto parámetro define si queremos que ignore las mayúsculas y minúsculas.

```
1. //propiedad: name
2. //value: Crys
3. //comienza en: 0
4. //sobre cualquier parte del valor del registro
5. //no toma en cuenta mayúsculas y minúsculas
6. this.store.find('name', 'Crys',0,true,false);
```

Lo que regresa el método “find” es el índice donde encuentra la primera coincidencia, en caso de no encontrar nada regresará un “-1”.

```
1. ,
2.
3. find: function(){
4. //tomamos lo que se introdujo en la caja de texto
5. var value = Ext.fly('txt').getValue();
6. //si no hay nada salimos de esta función
7. if(Ext.isEmpty(value)) return;
8. //realizamos la búsqueda sobre la propiedad “name”
9. var index = this.store.find('name',value,0,true,false);
10. //si en encontró algo
11. if(index>=0){
```

```

12. //tomamos el registro por medio del índice...
13. var record = this.store.getAt(index);
14. //e imprimimos la información encontrada
15. this.log(record.get('name')+' work as a '+record.get('occupation')+' and '+record.get('gender')
16. =='f'?she':he)'+ is '+record.get('age')+' years old');
17. }else{
18. //si nada fue encontrado se le avisa al usuario
19. this.log(''+value+' not found!');
20. }

```

Puedes ver que se ha utilizado el método “getAt” para tomar el registro completo dándole el índice que necesitamos.

Si sabemos el ID del registro podemos sacarlo inmediatamente utilizando el método “getById”, lo que vamos a hacer es verificar si el usuario introdujo un número en la caja de texto, de ser así utilizaremos el ID, si es texto entonces ejecutaremos el código anterior.

```

1. , //<---
2.
3. find: function(){
4. //tomamos lo que se introdujo en la caja de texto
5. var value = Ext.fly('txt').getValue();
6. //si no hay nada salimos de esta función
7. if(Ext.isEmpty(value)) return;
8. //si el valor es numérico
9. if(/^\d+$/.test(value)){
10. //buscamos por ID
11. var record = this.store.getById(value);
12. if(!Ext.isEmpty(record)){
13. //si se encontró algo se imprime
14. this.log(record.get('name')+' work as a '+record.get('occupation')+' and '+record.get('gend
er')=='f'?she':he)'+ is '+record.get('age')+' years old');
15. }else{
16. //si nada fue encontrado se avisa al usuario
17. this.log('Record with id: '+value+' was not found!');
18. }
19. }else{
20. //realizamos la búsqueda sobre la propiedad "name"
21. var index = this.store.find('name',value,0,true,false);
22. //si en encontró algo
23. if(index>=0){
24. //tomamos el registro por medio del índice...
25. var record = this.store.getAt(index);
26. //e imprimimos la información encontrada
27. this.log(record.get('name')+' work as a '+record.get('occupation')+' and '+record.get('gend
er')=='f'?she':he)'+ is '+record.get('age')+' years old');
28. }else{
29. //si nada fue encontrado se le avisa al usuario
30. this.log(''+value+' not found!');
31. }
32. }
33. }
34. }

```

El código anterior decide si la búsqueda será realidad por ID o por la propiedad especificada (en este caso “name”).

## Contar los registros del store

Para contar los registros que actualmente están en el store es muy fácil, únicamente utilizamos el método “getCount”.

```
1. ,
2.
3. count: function(){
4. //imprime el total de registros
5. this.log('Total records: '+this.store.getCount()+'');
6. }
```

Notar que este método solo regresa los registros que actualmente están en el store.

### El Log

Por último vamos a definir el método “log” que hemos estado usando para desplegar los mensajes.

```
1. ,
2.
3. log: function(txt){
4. var el = Ext.get('response'); // get the LOG
5. el.select('p.newest').removeClass('newest'); // quitar la última actualización
6. Ext.DomHelper.append(el,'<p class="newest">'+txt+'</p>'); //actualizar el log
7. el.scrollTo('top',el.dom.scrollHeight); //scroll abajo
8. el.select('p.newest').highlight('F5FC49',{duration:0.5}); //resaltar el ultimo mensaje
9. }
```

Lo que hicimos en el código anterior es tomar el nodo “response” y agregarle párrafos con el texto que recibe, luego hacemos que destelle en color amarillo.

### Conclusiones

Es importante que sepamos como buscar información contenida en un store, ya que este componente es muy usado para manipular información, es fundamental conocerlo para una mejor comprensión del Framework.

En este tema vimos un store muy sencillo que toma la información de un arreglo definido con JavaScript, en el mundo real la información viene de una base de datos, de una servicio Web o de algún otro lugar, en los siguientes temas veremos como realizar esto.

## Leer información de un XML

El día de hoy veremos como leer el contenido de un XML mediante un “reader” y lograr manipular la información mediante un Store, vamos a utilizar Ajax para cargar la información dentro del Store y trabajar con ella más adelante.

### Material de apoyo

Para continuar es necesario descargar el material de apoyo, descomprimir y copiar los tres archivos que contiene el zip a la carpeta “ajax” (la misma del tutorial anterior) que se encuentra en el servidor Web que instalamos en el primer capítulo del curso.

Para este tema vamos a utilizar el mismo código que escribimos en el tutorial anterior, esto quiere decir que haremos (visual y funcionalmente) lo mismo que en el tutorial anterior, la diferencia es que vamos a cambiar el Store para que pueda cargar la información mediante una llamada Ajax al servidor el cual regresará el contenido en formato XML.

No voy a explicar el código que viene en el material de apoyo pues ya lo hice el día de ayer, si tienes dudas sobre su funcionamiento te recomiendo repasar lo explicado anteriormente.

### Estructura general

Vamos a editar el archivo “xml.js” veremos algo semejante a esto:

```

1. //the namespace for this tutorial
2. Ext.ns('com.quizzpot.tutorial');
3.
4. com.quizzpot.tutorial.Store = {
5. init: function(){
6.
7. //we will write code only between the asterisks
8. //*****
9. //code here the record
10.
11. //code here the xml reader
12.
13. //code here the proxy
14.
15. //code here the Store
16.
17. //load the data with an Ajax call
18.
19. //*****
20. //the next code is the same as the last tutorial
21. //listeners for the buttons
22. Ext.fly('personBtn').on('click',this.find,this);
23. Ext.fly('txt').on('keyup',function(event,cmd){
24. if(event.getKey() === event.ENTER){ //when press ENTER
25. this.find(); // perform the search
26. }
27. },this);
28. Ext.fly('ascBtn').on('click',this.orderAsc,this);
29. Ext.fly('descBtn').on('click',this.orderDesc,this);
30. Ext.fly('older2030Btn').on('click',this.query,this);
31. Ext.fly('older30Btn').on('click',this.filter,this);
32. Ext.fly('countBtn').on('click',this.count,this);
33.
34. },
35.
36. orderAsc: function(){
37. this.store.sort('name','ASC'); // sort the store ASC
38. this.store.each(function(record){
39. this.log(record.get('name')); //print each name
40. },this);
41. this.log('_____');
42. },
43.
44. orderDesc: function(){
45. this.store.sort('name','DESC'); //sort the store DESC
46. this.store.each(function(record){
47. this.log(record.get('name')); //print each name
48. },this);
49. this.log('_____');
50. },
51.
52. filter: function(){
53. //filter people...
54. this.store.filterBy(function(record,id){
55. return record.get('age') >= 30; //older than 30 years old
56. });
57.
58. this.store.each(function(record){
59. //print on screen

```

```

60. this.log(record.get('name')+' is older than 30 '+ (record.get('gender')=='f'? 'she': 'he')+' is '+re
cord.get('age'));
61. },this);
62. //clear the filters
63. this.store.clearFilter();
64. this.log('_____');
65. },
66.
67. query: function(){
68. //query the store, search for people older than 20 and younger than 30
69. var collection = this.store.queryBy(function(record,id){
70. return record.get('age') >20 && record.get('age')<30;
71. });
72.
73. //for each item found...
74. collection.each(function(item,index){
75. //print the info on the screen
76. this.log(item.get('name')+' is '+item.get('age')+' and '+ (item.get('gender')=='f'? 'she': 'he')+' is
younger than 30');
77. },this);
78. this.log('_____');
79. },
80.
81. count: function(){
82. //count the records in the store
83. this.log('Total records: '+this.store.getCount()+'');
84. },
85.
86. find: function(){
87. var value = Ext.fly('txt').getValue();
88. if(Ext.isEmpty(value)) return;
89. //if the value is a number
90. if(/^\d+$/.test(value)){
91. //find by ID
92. var record = this.store.getById(value);
93. if(!Ext.isEmpty(record)){
94. //if found, log it
95. this.log(record.get('name')+' work as a '+record.get('occupation')+' and '+ (record.get('ge
nder')=='f'? 'she': 'he')+' is '+record.get('age')+' years old');
96. }else{
97. //alert the user if nothing is found
98. this.log('Record with id: '+value+' was not found!');
99. }
100. }else{
101. //if it is text, search the name property
102. var index = this.store.find('name',value,0,true,false);
103. //if something is found...
104. if(index>=0){
105. //get the record by the index...
106. var record = this.store.getAt(index);
107. //and print the information
108. this.log(record.get('name')+' work as a '+record.get('occupation')+' and '+ (record
.get('gender')=='f'? 'she': 'he')+' is '+record.get('age')+' years old');
109. }else{
110. //alert the user if nothing is found
111. this.log(''+value+' not found!');
112. }
113. }
114. },
115.

```

```

116. log: function(txt){
117. var el = Ext.get('response'); // get the LOG
118. el.select('p.newest').removeClass('newest'); // remove last update
119. Ext.DomHelper.append(el,'<p class="newest">'+txt+'</p>'); //update the log
120. el.scrollTo('top',el.dom.scrollHeight); //scroll down
121. el.select('p.newest').highlight('F5FC49',{duration:0.5}); //highlight the last message
122. }
123. }
124.
125. Ext.onReady(com.quizpot.tutorial.Store.init,com.quizpot.tutorial.Store);

```

El código anterior está debidamente empaquetado, y cuenta con métodos donde se invoca al store, el cual no hemos definido aún, por lo tanto si en este momento das clic sobre cualquier botón en el HTML aparecerá un error. Para este tutorial vamos a escribir el código solo entre los asteriscos que se encuentran al inicio, he puesto comentarios de lo que vamos a ir realizando paso a paso.

### El XML a utilizar

Ext JS nos proporciona una manera muy fácil de manipular la información contenida en un XML, no tenemos que movernos por el árbol generado para acceder a las propiedades y atributos ni recorrer los nodos mediante ciclos, esto es una gran ventaja pues nos ahorra mucho tiempo en el desarrollo.

El XML que vamos a utilizar para este tutorial esta contenido en el archivo “data.php” donde únicamente se están modificando las cabeceras para la respuesta e imprimiendo la información en XML, en el mundo real la información saldría de una base de datos, de un servicio Web o de alguna otra fuente, por cuestiones de aprendizaje lo he dejado así de simple:

```

1. <?php
2. header("Content-Type: text/xml");
3.
4. echo '<?xml version="1.0" encoding="UTF-8"?>';
5. ?
6.
7. <dataset>
8. <results>9</results>
9. <person active="true">
10. <id>1</id>
11. <name>Crysfel</name>
12. <occupation>Software developer</occupation>
13. <gender>m</gender>
14. <age>25</age>
15. </person>
16. <person active="false">
17. <id>2</id>
18. <name>Sasha</name>
19. <occupation>Figure skater</occupation>
20. <gender>f</gender>
21. <age>24</age>
22. </person>
23. <person active="true">
24. <id>3</id>
25. <name>Jack</name>
26. <occupation>Software Architect</occupation>
27. <gender>m</gender>
28. <age>35</age>
29. </person>
30. <person active="true">
31. <id>4</id>
32. <name>John</name>
33. <occupation>JavaScript developer</occupation>
34. <gender>f</gender>

```

```

35. <age>24</age>
36. </person>
37. <person active="true">
38. <id>5</id>
39. <name>Sara</name>
40. <occupation>Designer</occupation>
41. <gender>f</gender>
42. <age>31</age>
43. </person>
44. <person active="true">
45. <id>6</id>
46. <name>Nicole</name>
47. <occupation>Tester</occupation>
48. <gender>f</gender>
49. <age>28</age>
50. </person>
51. <person active="false">
52. <id>7</id>
53. <name>Carl</name>
54. <occupation>Photographer</occupation>
55. <gender>m</gender>
56. <age>45</age>
57. </person>
58. <person active="true">
59. <id>8</id>
60. <name>Will</name>
61. <occupation>Actor</occupation>
62. <gender>m</gender>
63. <age>32</age>
64. </person>
65. <person active="false">
66. <id>9</id>
67. <name>Penny</name>
68. <occupation>Waitress</occupation>
69. <gender>f</gender>
70. <age>29</age>
71. </person>
72. </dataset>

```

A continuación se explican los pasos que son necesarios para poder manipular la información contenida en el XML.

### Paso 1: Crear el registro Person

Lo primero que debemos hacer es definir los campos que tendrán los registros, además necesitamos indicarle a cada registro de donde proviene el contenido que tendrá.

```

1. //create the "Person" record
2. var Person = Ext.data.Record.create([
3. {name: 'active', mapping:'@active', type:'boolean'}, // mapping an attribute and setting a type for
this field
4. {name: 'name', mapping: 'name'},// "mapping" property not needed if it's the same as "name"
5. {name: 'occupation'}, // This field will use "occupation" as the mapping.
6. {name: 'age', type:'float'}, // this field will use "age" as the mapping and its a float type
7. {name: 'gender'}
8.]);

```

Para crear un registro se utiliza el método “create” del objeto “Ext.data.Record”, el cual recibe un arreglo con los campos que contendrá, mediante la propiedad “name” se le indica el nombre que se le dará al campo, también se le puede especificar el tipo de dato que será (boolean, date, float, etc...) y de ser necesario utilizar la propiedad “mapping” para hacer la relación con la información en el XML.

Algo que quiero resaltar es que la propiedad “active” tiene un “mapping” el cual ha sido relacionado con un atributo que está en el nodo “person” del XML, hay que notar que para indicar que es un atributo y no un nodo es necesario anteponer una arroba (@).

### Paso 2: Crear el “reader” para XML

Una vez que definimos las propiedades del registro necesitamos crear el “reader” que se encargará de manipular el XML.

```
1. //creates the reader for the XML data
2. var reader = new Ext.data.XmlReader({
3. totalRecords: "results", // The element which contains the total dataset size (optional)
4. record: "person", // The repeated element which contains row information
5. id: "id" // The element within the row that provides an ID for the record (optional)
6. }, Person);
```

El constructor del “XmlReader” recibe como primer parámetro un objeto de configuración en el cual se define el nodo donde se encuentra el total de registros mediante la propiedad “totalRecords”, ese nodo debe estar en el XML; se define también el “id” que será utilizado para los registros y lo más importante, la propiedad “record” que es el nodo del XML de donde toma la información para llenar los registros, es importante mencionar que en esta propiedad es donde en el paso uno de este tutorial, se definieron las propiedades de los registros.

Como segundo parámetro recibe el registro que será utilizado y llenado con la información.

### Paso 3: Crear el Proxy

Si queremos que mediante llamadas Ajax se cargue la información dentro del store tenemos que definir el lugar de donde se solicitará la información y opcionalmente el método HTTP usado para realizar la llamada Ajax (GET o POST).

```
1. //creates the proxy
2. var proxy = new Ext.data.HttpProxy({
3. method:'POST', //configure the http method GET or POST
4. url: 'data.php' //the URL for the Ajax call
5. });
```

El código anterior define el método usado y la URL que será invocada.

### Paso 4: Crear el Store

Después de definir el “reader” y el “proxy” podemos crear el store de la siguiente manera:

```
1. //creates the Ext.data.Store
2. this.store = new Ext.data.Store({
3. proxy: proxy, //setting the proxy
4. reader: reader //setting the reader
5. });
```

Con eso es suficiente por ahora, es necesario aclarar que el Store acepta diferentes implementaciones de “readers”, por ejemplo el JsonReader o en este caso XmlReader.

### Paso 5: Cargar la información en el Store

Por último vamos a solicitar mediante Ajax la información al servidor de la siguiente manera:

```
1. //loading the data
2. this.store.load({params:{param1:'value'}});
```

El método “load” realiza la petición al servidor utilizando Ajax, opcionalmente podemos pasarle los parámetros que necesitemos mediante un objeto que contenga la propiedad “params”, esto es muy útil si la información que solicitamos es variante, por ejemplo aquí podría enviarle al servidor el “id” de una persona para que sólo ese registro sea regresado. A continuación voy a poner un mensaje de espera

para avisarle al usuario que la información se está cargando, como la carga es muy rápida voy a ponerle un “delay” para que veamos por más tiempo el mensaje.

```
1. //loading the data
2. Ext.Msg.wait('Loading... please wait!', 'Wait');
3. this.store.load({params:{param1:'value'}});
4. this.store.on('load',function(){
5. //delay the message 2 seconds
6. setTimeout(function(){
7. Ext.Msg.hide(); // just to see the waiting message XD (don't do it in the real world)
8. },2000);
9. });
```

Nota que he creado un “listener” para el evento “load” del store, donde definí una función que será ejecutada tan pronto como la información sea cargada en el store, dentro de esta función esconde el mensaje.

### Conclusiones

En este tema además de mostrar como manipular un XML también se demostró la modularidad del Framework, si lo notaste, el código del tutorial anterior no fue modificado en lo absoluto, esto es por que estamos usando la misma información pero ahora esta contenida en un XML, así que simplemente se cambia la fuente de información y el resto de la aplicación sigue funcionando como si nada pasara, estas son una de las ventajas de utilizar Ext JS.

## Leer información en formato JSON

El tema de hoy muestro como poder manipular información en formato JSON mediante un Store y por medio de Ajax será solicitada al servidor para ser procesada localmente.

### Material de apoyo

El ejercicio que haremos en este tutorial será igual a los dos tutoriales anteriores, únicamente cambiaremos la fuente de datos, así que vamos a descargar el material de apoyo, lo descomprimimos y copiamos los archivos dentro de la carpeta “ajax” que creamos al inicio de este capítulo.

### La información

La información que vamos a utilizar está contenida en formato JSON de la siguiente manera:

```
1. <?php
2. header("Content-Type: text/plain");
3.
4. echo "{"
5. total:9,
6. data:[{
7. id: 1,
8. name: 'Crysfel',
9. occupation: 'Software developer',
10. gender: 'm',
11. age: 25
12. },{
13. id: 2,
14. name: 'Sasha',
15. occupation: 'Figure skater',
16. gender: 'f',
17. age: 24
18. },{
19. id: 3,
20. name: 'Jack',
21. occupation: 'Software Architect',
22. gender: 'm',
```

```

23. age: 35
24. },{
25. id: 4,
26. name: 'John',
27. occupation: 'Javascript developer',
28. gender: 'm',
29. age: 22
30. },{
31. id: 5,
32. name: 'Sara',
33. occupation: 'Designer',
34. gender: 'f',
35. age: 31
36. },{
37. id: 6,
38. name: 'Nicole',
39. occupation: 'Tester',
40. gender: 'f',
41. age: 31
42. },{
43. id: 7,
44. name: 'Carl',
45. occupation: 'Photographer',
46. gender: 'm',
47. age: 45
48. },{
49. id: 8,
50. name: 'Will',
51. occupation: 'Actor',
52. gender: 'm',
53. age: 32
54. },{
55. id: 9,
56. name: 'Penny',
57. occupation: 'Waitress',
58. gender: 'f',
59. age: 28
60. }]
61. }";
62. ?>

```

Este código se encuentra en el archivo “jsondata.php” que viene en el material de apoyo, es importante mencionar que por cuestiones de simplicidad la información está escrita directamente en el código, pero en el mundo real vendría de una base de datos, un servicio Web o algún otro lugar.

### Creando el Store

Podemos crear un Store que pueda manipular la información en formato JSON de dos maneras, una es haciendo lo mismo que el tutorial anterior únicamente cambiando el “reader” de XML por uno que lea JSON y definirle la propiedad “root” que es donde se encuentran los registros.

```

1. //create the "Person" record
2. var Person = Ext.data.Record.create([
3. {name: 'name', mapping: 'name'},// "mapping" property not needed if it is the same as "name"
4. {name: 'occupation'}, // This field will use "occupation" as the mapping.
5. {name: 'age', type:'float'}, // this field will use "age" as the mapping and its a float type
6. {name: 'gender'}
7.]);
8.
9. //creates the reader for the JSON data
10. var reader = new Ext.data.JsonReader({

```

```

11. totalProperty: 'total', // The element which contains the total dataset size (optional)
12. root: 'data', // The repeated element which contains row information
13. id: 'id' // The element within the row that provides an ID for the record (optional)
14. }, Person);
15.
16. //creates the proxy
17. var proxy = new Ext.data.HttpProxy({
18. method:'POST', //configure the http method GET or POST
19. url: 'jsondata.php' //the URL for the ajax call
20. });
21.
22. //creates the Ext.data.Store
23. this.store = new Ext.data.Store({
24. proxy: proxy, //setting the proxy
25. reader: reader //setting the reader
26. });

```

Como puedes ver, el código es muy semejante al ejemplo que vimos con XML en el tema anterior, solo han cambiado unas pocas líneas. La segunda alternativa es utilizar el objeto “JsonStore”, el cual ya tiene incluido un lector para JSON así como un Proxy para realizar las peticiones Ajax al servidor, utilizando este método podemos ahorrarnos muchas líneas de código reemplazando todo el código anterior por lo siguiente:

```

1. this.store = new Ext.data.JsonStore({
2. url: 'jsondata.php',
3. root: 'data',
4. fields: ['name','occupation','gender',{name:'age',type:'float'}]
5. });

```

La propiedad “url” es donde se solicitará la información mediante Ajax, la propiedad “root” es donde se encuentran los registros que se utilizarán para este store y la propiedad “fields” es un arreglo donde especificamos los campos de los registros que recibiremos, esto lo hacemos únicamente definiendo el nombre de la propiedad o bien utilizando un objeto donde podemos definir el nombre, el mapping o el tipo de información que tendrá esa propiedad.

### **El resto del código**

El resto del código lo he explicado en los temas anteriores, básicamente lo único que hacemos es agregar un “listener” al evento clic de los botones que se encuentra en el documento HTML para luego filtrar los registros por los criterios necesarios.

### **Conclusiones**

El formato JSON es muy sencillo de manipular, pues crear un store para este formato de información no es complicado y lo hacemos en unas cuantas líneas de código. Si has notado, el Store es un componente muy importante del Framework, además de que ha sido planeado para soportar diferentes formatos de información. En lo que resta del curso vamos a estar utilizando este componente en los formularios, las grillas, los árboles y varios componentes más que requieran manipular información.

## Quizz: Ajax y el objeto Store

En este Quizz se evalúan varios aspectos sobre las peticiones al servidor a través de Ajax y como Ext JS nos ayuda con esta tarea, además se evaluará el componente Store. ¡Suerte!

### 1.- ¿Se pueden crear instancias del componente “Ext.Ajax”?

- Si, para cada petición hay que crear una instancia.
- No, porque es un Singleton.
- Si, porque de lo contrario no se puede realizar ninguna petición.
- No sé

### 2.- ¿Cuál es la mejor manera de mandar parámetros al servidor con el componente “Ext.Ajax”?

- Agregándoselos en la URL : servlet.do?param1=23¶m2=10
- Utilizando la propiedad “params” en la configuración.
- No se pueden mandar parámetros al servidor.

### 3.- La propiedad “params” ¿qué formatos acepta?

- Un Objeto de JavaScript con los parámetros a enviar: {param: ‘value’}
- Un String con los parámetros: “param=value”
- Una función que retorne los parámetros: function(){ return “param=2”;}
- Todas las anteriores.

### 4.- ¿Cuáles son los “métodos” con los que podemos realizar las peticiones al servidor?

- GET y POST.
- Solamente mediante GET.
- GET, POST, PUT y DELETE.
- Solamente mediante POST.

### 5.- ¿Cuándo es invocada la función configurada en la propiedad “success”?

- Cuando el servidor regresa la información correctamente con “status” 200.
- Cuando no ha sucedido ningún error en la comunicación.
- Cuando todo ha salido bien.
- Siempre se ejecuta esta función.

### 6.- ¿Cuándo se ejecuta la función configurada en la propiedad “failure”?

- Cuando el servidor manda un estatus 404.
- Cuando el servidor regresa un estatus 403.
- Cuando se ocasiona un error 500.
- Cuando se produce un error en el servidor o en la conexión y no regresa el “status” 200.

**7.- ¿Qué es un Store?**

- Es un componente que almacena temporalmente información mediante registros y es utilizado como caché.
- Es una base de datos alojada en el servidor.
- Es un arreglo donde están alojados registros cuando navegamos en un sistema.
- Es un componente capaz de interpretar cualquier cantidad de información y desplegarla en pantalla.

**8.- ¿Cuál es el componente que necesita el store para poder interpretar la información que recibe?**

- Un “Ext.data.XmlReader” que maneje correctamente la información.
- Un “Ext.data.Reader” que maneje cualquier tipo de información.
- Una implementación del componente abstracto “Ext.data.DataReader” con el formato deseado (XmlReader, JsonReader, ArrayReader).
- Un “Ext.data.JsonReader” solamente.

**9.- Para cargar información local contenida en un arreglo, ¿qué método utilizamos?**

- Store.load()
- Store.loadData(array)
- Store.loadArray(array)
- Store.load(array)

**10.- Si necesitamos mandar parámetros adicionales al servidor utilizando un “Store”, ¿cómo lo hacemos?**

- Store.load({param1:'value',param2:'value'});
- Store.load("params1=value¶m2=value");
- Store.load({params:{param1:'value',param2='value'}});
- Store.loadData("params1=value¶m2=value");

**11.- Dado el siguiente código, ¿que se imprime en la consola de Firebug? (supón que si existe un registro para el criterio de búsqueda dado).**

```
var result = store.find('name', 'crysfel');
console.debug(result);
```

- Imprime todos los campos del registro encontrado (lastname, age, etc.).
- Imprime el índice donde se encuentra el registro.
- Imprime un arreglo con los registros encontrados para ese criterio de búsqueda
- Imprime la palabra “crysfel”.

**12.- ¿Cuál es el resultado al ejecutar el siguiente código?**

```
var collection = store.queryBy(function(record,id){
 return record.get('age') >20 && record.get('age')<30;
});
```

- La variable “collection” contiene los registros que no están entre 20 y 30.

- La variable “collection” contiene todos los registros cuya propiedad “age” es de 20 a 30.
- La variable “collection” contiene todos los registros cuya propiedad “age” es de 21 a 29.
- La variable “collection” contiene un “true” o “false” dependiendo si existen registros con el criterio dado.

# Paneles, Ventanas y Pestañas

El panel es uno de los componentes mas utilizados en el framework es de importancia comprender su funcionamiento, las ventanas son útiles para almacenar otros componentes.

## ¿Que es un panel, para que sirve y cómo se crean?

El panel es un componente básico dentro del Framework de Ext JS ya que muchos componentes heredan de éste, es por eso que conocer su funcionamiento es fundamental.

### Material de apoyo

Antes de continuar es necesario descargar el material de apoyo, descomprimirlo y copiar los archivos al servidor Web que instalamos en el primer capítulo de este curso, dentro de la carpeta “curso”, en la cual hemos estado trabajando; vamos a crear una carpeta que se llame “panels” y pegamos el material de apoyo.

### ¿Qué es un panel?

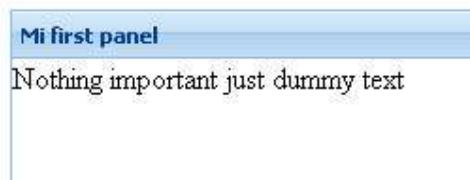
Un panel es un contenedor con funcionalidades específicas que nos permiten construir perfectamente interfaces de usuario, entre algunas características podemos agregarle barras de herramientas superiores e inferiores, botones en la parte inferior o hacer que pueda colapsarse. Los paneles pueden ser fácilmente asignados a cualquier otro contenedor.

### Crear un primer panel

Para crear un panel es muy sencillo, basta con hacer lo siguiente:

```
1. var main = new Ext.Panel({
2. title: 'My first panel', //el título del panel
3. width:250, //la anchura del panel
4. height:300, //la altura que tendrá
5. renderTo: 'frame', //el elemento donde será insertado
6. html: 'Nothing important just dummy text' //el contenido del panel
7. });
```

El código anterior crea un panel con el título de “My first panel”, con dimensiones de 250 por 300 píxeles y lo colocará dentro del elemento “frame” que se encuentra en el HTML, el contenido es solamente el texto “Nothing important just dummy text”.



Ejemplo básico de un panel

Lo más importante en el código anterior es notar la propiedad “renderTo”, esta propiedad acepta el “id” del nodo en el DOM en el cual será insertado el panel que estamos creando.

En el siguiente ejemplo se muestra otra alternativa por si necesitamos “renderizar” el panel en algún otro momento.

```
1. var main = new Ext.Panel({
2. title: 'My first panel',
3. width:250,
```

```

4. height:300,
5. html: 'Nothing important just dummy text'
6. });
7. //usamos el método "render" para imprimir el panel en la pantalla
8. main.render('frame');

```

Como se puede observar el objeto “Panel” cuenta con un método capaz de “renderizar” o insertar el HTML necesario para que el usuario pueda mirar el panel en la pantalla, este método se llama “render” y acepta como parámetro el “id” del nodo en el DOM donde será insertado.

### **El contenido del panel**

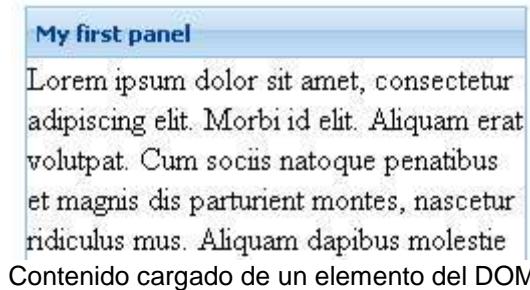
El contenido de un panel puede ser asignado de cuatro maneras diferentes, hemos visto una de ellas en los códigos anteriores, usando la propiedad “html” se le puede especificar el contenido que se necesite agregar.

Una segunda opción es definir mediante la propiedad “contentEl” el “id” del elemento que necesitemos meter al panel, es necesario que este elemento ya se encuentre en el DOM para que pueda ser insertado en el panel, vamos a descomentar el código HTML que viene en el material de apoyo y luego escribimos el siguiente código.

```

1. var main = new Ext.Panel({
2. title: 'My first panel',
3. width:250,
4. height:300,
5. contentEl: 'content' //usamos un elemento del DOM como contenido
6. });
7. main.render('frame');

```

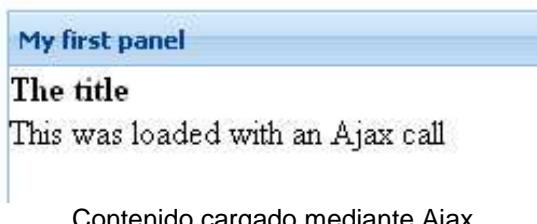


La tercera opción consiste en realizar una llamada Ajax para solicitar el contenido al servidor e insertar dentro del panel la respuesta obtenida, es importante mencionar que el servidor debe enviar código HTML para que sea “renderizado” correctamente en el navegador.

```

1. var main = new Ext.Panel({
2. title: 'My first panel',
3. width:250,
4. height:300
5. });
6. main.render('frame');
7. //usando Ajax para insertar el contenido
8. main.load('panel.php');

```



También podemos utilizar la propiedad “autoLoad”, de esta manera cuando creamos la instancia automáticamente se solicitará el contenido mediante una llamada Ajax.

```
1. var main = new Ext.Panel({
2. title: 'Mi primer panel',
3. width:250,
4. height:300,
5. autoLoad: 'panel.php' //<--- El contenido será sacado de aquí automáticamente
6. });
7. main.render('frame');
```

La cuarta opción es insertar otros componentes de Ext JS, pueden ser Paneles, Tabs o pestañas, árboles, acordeones, formularios, etc. Para hacer esto primero necesitamos crear el componente que necesitemos agregar y después asignárselo mediante la propiedad “items” que es un arreglo de componentes.

```
1. //Creamos el panel interior
2. var panel1 = new Ext.Panel({
3. title: 'Users',
4. html: 'The content',
5. bodyStyle: 'padding:10px;', //Podemos asignarle estilos al div contenedor
6. height:200,
7. border: false //Le podemos quitar el borde al panel
8. });
9.
10. //El panel principal que contendrá otros paneles dentro de si
11. var main = new Ext.Panel({
12. title: 'My first panel',
13. width:250,
14. height:600,
15. items: [panel1] //Aqui se le asignan componentes que contendrá
16. });
17. //Se renderizan todos los paneles creados
18. main.render('frame');
```

En el código anterior se creó un panel y después se insertó dentro de otro, esto es muy común cuando desarrollamos layouts con Ext JS, es importante mencionar que al llamar el método “render” del panel principal, automáticamente “renderiza” todos los componentes que contenga.



Componentes dentro de un panel

### Colapsar los paneles

Una de las características que tiene los paneles es que pueden ser colapsados mediante un botón en la parte superior derecha del panel, para realizarlo simplemente se le asigna “true” a la propiedad “collapsible”.

```
1. //Creamos el panel interior
2. var panel1 = new Ext.Panel({
3. title: 'Users',
4. html: 'The content',
5. bodyStyle: 'padding:10px;',
```

```

6. height:200,
7. border: false,
8. collapsible: true //mediante esta propiedad el panel se colapsa
9. });
10.
11. //... no se muestra el resto del código por cuestiones de aprendizaje

```



Panel colapsable

También es posible que al dar clic sobre cualquier parte del título el panel se colapse, esto lo logramos asignando “true” a la propiedad “titleCollapse” de la siguiente manera:

```

1. //Creamos el panel interior
2. var panel1 = new Ext.Panel({
3. title: 'Users',
4. html: 'The content',
5. bodyStyle: 'padding:10px;',
6. height:200,
7. border: false,
8. collapsible: true,
9. titleCollapse: true //ahora se colapsará dando clic sobre el título
10. });
11.
12. //... no se muestra el resto del código por cuestiones de aprendizaje

```

### Agregar un ícono en el título del panel

Es común que alguna vez necesitemos agregar íconos en la barra de título para realizar un buen “look and feel”, para hacerlo solo se necesita definir una clase de CSS la cual pondrá como “background” la imagen que usaremos como ícono.

```

1. .users{
2. background: url(icons/users.png) 0px 0px no-repeat !important;
3. }

```

Una vez definida la clase CSS se le debe asignar al panel mediante la propiedad “iconCls” de la siguiente manera:

```

1. //Creamos el panel interior
2. var panel1 = new Ext.Panel({
3. title: 'Users',
4. html: 'The content',
5. bodyStyle: 'padding:10px;',
6. height:200,
7. border: false,
8. collapsible: true,
9. titleCollapse: true,
10. iconCls: 'users' //con esto se le agregará el ícono deseado
11. });
12.
13. //... no se muestra el resto del código por cuestiones de aprendizaje

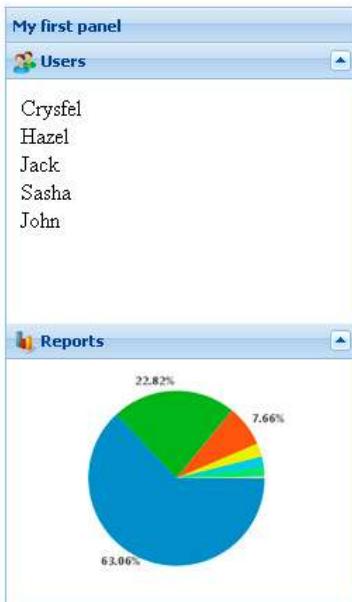
```



Panel con ícono personalizado

### La propiedad “defaults”

Vamos a crear otros dos paneles exactamente iguales al anterior, solo que le vamos a cambiar el título y el ícono que tendrán.



```

1. var panel1 = new Ext.Panel({
2. title: 'Users',
3. iconCls: 'users',
4. collapsible:true,
5. border: false,
6. bodyStyle: 'padding:10px;',
7. titleCollapse: true,
8. height:200
9. });
10.
11. var panel2 = new Ext.Panel({
12. title: 'Reports',
13. iconCls: 'reports',
14. collapsible:true,
15. border: false,
16. bodyStyle: 'padding:10px;',
17. titleCollapse: true,
18. height:200
19. });
20.
21. var panel3 = new Ext.Panel({
22. title: 'Documents',
23. iconCls: 'documents',
24. collapsible:true,
25. border: false,
26. bodyStyle: 'padding:10px;',
27. titleCollapse: true,
28. height:200
29. });
30.
31. var main = new Ext.Panel({
32. title: 'My first panel',
33. width:250,
34. height:600,
35. items: [panel1,panel2,panel3]
36. });
37. main.render('frame');
```

Si has notado hay mucho código que se repite cuando creamos los primeros tres paneles, exactamente cinco propiedades que no varían en lo absoluto, Ext JS nos permite definir propiedades por defecto para los componentes que utilizan las mismas propiedades, de esta manera evitamos escribir código de más; mediante la propiedad “defaults” podemos definir las propiedades que queramos que sean aplicadas a los componentes que contiene el componente padre.

```

1. var panel1 = new Ext.Panel({
2. title: 'Users',
3. iconCls: 'users'
4. });
5. var panel2 = new Ext.Panel({
6. title: 'Reports',
7. iconCls: 'reports'
8. });
9. var panel3 = new Ext.Panel({
10. title: 'Documents',
11. iconCls: 'documents'
12. });
13.
14. var main = new Ext.Panel({
15. title: 'My first panel',
16. width:250,
17. height:600,
18. defaults: { // con esto evitamos...
19. collapsible:true, //duplicar código...
20. border: false, // y todas estas...
21. bodyStyle: 'padding:10px;', // propiedades ...
22. titleCollapse: true, // son agregadas a...
23. height:200 //todos los paneles internos
24. },
25. items: [panel1,panel2,panel3]
26. });
27. main.render('frame');

```

Varios panels dentro de un panel principal

### Cargar la información

Hemos visto como cargar el contenido de los paneles utilizando Ajax, lo que vamos a realizar en este momento es pasarle parámetros para que nos regrese la información correcta, para esto en el método “load” del panel le pasamos un objeto a la “url”, el método a usar y los parámetros que necesitemos.

```

1. panel1.load({
2. url: 'panel.php',
3. method: 'GET',
4. params: {data:'users'}
5. });
6. panel2.load({
7. url: 'panel.php',
8. method: 'GET',
9. params: {data:'report'}
10. });
11. panel3.load({
12. url: 'panel.php',
13. method: 'GET',
14. params: {data:'documents'}
15. });

```

Todas las peticiones son realizadas al mismo archivo de PHP el cual recibe un parámetro y dependiendo de su contenido nos regresa la información adecuada. Producto final

### Conclusiones

En este tema vimos como utilizar el Panel, vimos también algunas propiedades y métodos de los más comunes así que te recomiendo mirar el API y jugar un poco con este componente. En el siguiente capítulo voy a mostrar como realizar diferentes layouts para generar diferentes interfaces, por ahora hemos realizado una especie de acordeón muy semejante a la que utiliza el Microsoft Outlook en la parte izquierda.

## Una ventana flotante

Las ventanas son muy útiles para desplegar formularios o información que nos interese, en este tema veremos algunas propiedades y métodos importantes para usar ventanas en nuestras aplicaciones.

### Material de apoyo

Para continuar es necesario descargar el material de apoyo, donde únicamente viene un HTML que incluye la librería de Ext JS, y un JS que está vacío.

### Empaquetando

Es una buena práctica “empaquetar” el código que escribiremos, ya hemos discutido las razones anteriormente.

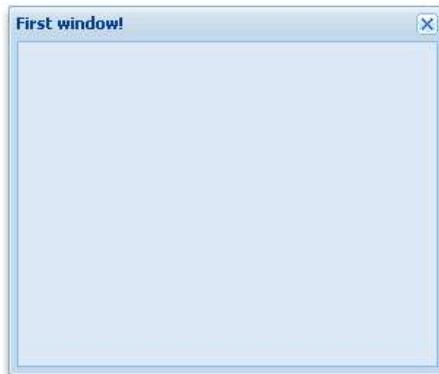
```
1. //the namespace for this tutorial
2. Ext.ns('com.quizzpot.tutorial');
3.
4. //the blank image
5. Ext.BLANK_IMAGE_URL = './ext-2.2/resources/images/default/s.gif';
6.
7. com.quizzpot.tutorial.Window = {
8. init: function(){
9. //code goes here
10. }
11. }
12.
13. Ext.onReady(com.quizzpot.tutorial.Window.init,com.quizzpot.tutorial.Window);
```

Escribiremos el resto del código dentro de la función “init”, de esta manera evitamos futuros problemas.

### Una ventana básica

El componente “Ext.Window” hereda los métodos y propiedades del componente “Ext.Panel” esto quiere decir que las propiedades que usamos para los paneles también las podemos utilizar para las ventanas, además podemos utilizar algunas otras propiedades que son únicas del componente Window. Para crear una ventana necesitamos únicamente 3 propiedades de configuración, el título, el ancho y el alto.

```
1. var win = new Ext.Window({
2. title: 'First window!', //the title of the window
3. width: 300,
4. height:250
5. });
6. //displaying the window to the user
7. win.show();
```



Ventana básica

El código anterior muestra como crear una ventana en su forma más básica, es importante mencionar que una vez que hemos creado una “instancia” del componente “Window” necesitamos invocar el método “show” para que la ventana aparezca en pantalla.

### Otras configuraciones

Por defecto las ventanas aparecen al centro de la pantalla, podemos modificar su posición especificándole la propiedad x,y; también podemos hacer que la ventana sea modal, es decir que todo el fondo sea cubierto por una capa transparente color gris permitiendo enfocar la atención únicamente en la ventana, también podemos hacerla maximizable y minimizable configurando las propiedades que se muestran a continuación.

```
1. var win = new Ext.Window({
2. title: 'First window!',
3. width: 300,
4. height:250,
5. minimizable: true, //show the minimize button
6. maximizable: true, //show the maxmize button
7. modal: true, //set the Window to modal
8. x: 100, //specify the left value of the window
9. y: 100 //specify the top value of the window
10.});
11. //display the window in the screen
12. win.show();
```



Botón minimizar, maximizar y es modal

### Minimizar una ventana

Es importante mencionar que se debe implementar la funcionalidad del botón minimizar ya que las ventanas no lo tienen definido por defecto, esto nos da la libertad de implementar dicha funcionalidad dependiendo de nuestras necesidades. Para lograrlo vamos a poner un “listener” al evento “minimize” de la siguiente manera:

```
1. var win = new Ext.Window({
2. title: 'Quizzpot.com',
3. width: 500,
4. height:350,
5. minimizable: true,
6. maximizable: true,
7. html: '<iframe src="http://www.quizzpot.com" style="width:100%;height:100%;border:none;"></if
rame>'
8.});
9. win.show();
10.
11. //fire when the user clicks the minimize button
```

```
12. win.on('minimize',function(w){
13. console.debug('minimizando...');
14. w.collapse(); //collapse the window
15. });
```

El código anterior crea un “listener” que se ejecuta cuando el usuario da click en el botón minimizar de la ventana, en ese momento se dispara la función definida que únicamente colapsa la ventana, hay que aclarar que cada quien hace lo que necesite o requiera dentro de esta función.

## Contenido

Para asignarle contenido a una ventana se hace de la misma forma que los paneles:

- Usando la propiedad “html” y escribiendo el html directamente.
- Usando la propiedad “contentEl” tomará un “id” de un elemento previamente cargado en el HTML y se lo asignará como contenido a la ventana.
- Usando la propiedad “autoLoad” o el método “load” para cargar el contenido utilizando Ajax.
- Utilizando la propiedad “items” y asignándole un arreglo de componentes Ext JS.

Ya hemos estudiado más a detalle los puntos anteriores, te recomiendo darle un repaso al tema anterior (paneles) en caso de tener dudas.

## Cargando sitios externos

Si necesitamos desplegar algún sitio externo como Google o Yahoo, podemos utilizar un iFrame para cargar el sitio, esto lo hacemos utilizando la propiedad “html” de la siguiente manera.

```
1. var win = new Ext.Window({
2. title: 'First window!',
3. width: 500,
4. height:350,
5. minimizable: true,
6. maximizable: true,
7. html: '<iframe src="http://www.quizzpot.com" style="width:100%;height:100%;border:none;"></if
rame>'
8. });
9.
10. var win2 = new Ext.Window({
11. title: 'First window!',
12. width: 500,
13. height:350,
14. minimizable: true,
15. maximizable: true,
16. html: '<iframe src="http://www.google.com" style="width:100%;height:100%;border:none;"></if
ame>'
17. });
18.
19. win.show();
20. win2.show();
```



Ventana con sitios externos

### Conclusiones

En este tema se puede observar como se utiliza la herencia correctamente, ya que una ventana es prácticamente un panel con funcionalidad extras como Drag, botones de maximizar y minimizar, además cuenta con los mismos métodos que el panel para asignarle contenido.

## Barras de herramientas en paneles y ventanas

### Material de apoyo

Para continuar es necesario descargar el material de apoyo, descomprimir y copiar el contenido a la carpeta "panels" que creamos al inicio de este capítulo, la cual está dentro de la carpeta "curso" en el servidor Web que hemos instalado en el primer capítulo. El material de apoyo es un HTML que importa al Framework de Ext JS y además tiene definidas unas clases de CSS que utilizaremos más adelante, hay un archivo JS en blanco, dentro de este vamos a escribir el código del tutorial y también viene una carpeta con iconos que vamos a usar para los botones. El ejercicio que haremos será simular un explorador Web, utilizando una barra de herramientas para los botones necesarios, vamos a usar un iFrame para desplegar las páginas que mediante la barra de herramientas le estaremos indicando.

### Ext JS 3.0

A partir de este tema vamos a utilizar la versión 3.0 del Framework de Ext JS, así que si no lo tienes hay que descargarlo y copiarlo a nuestro servidor Web dentro de la carpeta "curso" donde estamos alojando los ejemplos de este curso.

### Empaquetando el tutorial

Vamos a empaquetar el código del tutorial para evitar conflictos en el futuro.

1. //the namespace for this tutorial
2. Ext.ns('com.quizzpot.tutorial');
3. //the blank image
4. Ext.BLANK\_IMAGE\_URL = '../ext-3.0-rc1/resources/images/default/s.gif';
- 5.
6. com.quizzpot.tutorial.ToolbarTutorial = {
7. init: function(){
8. //code goes here
9. }
10. }
11. Ext.onReady(com.quizzpot.tutorial.ToolbarTutorial.init, com.quizzpot.tutorial.ToolbarTutorial);

## Barra de herramientas

Vamos a crear una ventana que contenga una barra de herramientas en la parte superior de la siguiente manera:

```
1. this.win = new Ext.Window({
2. title: 'Quizzpot Explorer',
3. width: 600,
4. height:450,
5. tbar: [// <--- ToolBar
6. {text:'Back'}, // <--- Buttons
7. {text:'Forward'},
8. {text:'Reload'},
9. {text:'Stop'},
10. {text:'Home'}
11.],
12. maximizable: true,
13. maskDisabled: true,
14. bodyStyle: 'background-color:#fff',
15. html: '<iframe id="container" src="http://www.google.com" style="width:100%;height:100%;border:none"></iframe>'
16.});
17.
18. this.win.show();
```

El código anterior crea una ventana con un iFrame donde muestra la página de Google, además tiene algunas otras propiedades que ya las hemos estudiado en temas anteriores por lo tanto no las explicaré ahora, en este tema quiero concentrarme en la propiedad “tbar” la cual es la responsable de asignarle al panel (recuerda que “Window” hereda las propiedades del “Panel”) una barra de herramientas en la parte superior, la cual contiene botones en este caso.



Barra de herramientas básica

La propiedad “tbar” acepta un arreglo de botones o un objeto de configuración para el componente “Ext.Toolbar” o una instancia del componente “Ext.Toolbar”, por lo tanto el código anterior lo podemos escribir también de la siguiente manera:

```
1. this.win = new Ext.Window({
2. title: 'Quizzpot Explorer',
3. width: 600,
```

```

4. height:450,
5. tbar: {
6. items: [
7. {text:'Back'},
8. {text:'Forward'},
9. {text:'Reload'},
10. {text:'Stop'},
11. {text:'Home'}
12.]
13. },
14. maximizable: true,
15. maskDisabled: true,
16. bodyStyle: 'background-color:#fff',
17. html: '<iframe id="container" src="http://www.google.com" style="width:100%;height:100%;border:none"></iframe>'
18. });
19.
20. this.win.show();

```

En el código anterior la propiedad “tbar” recibe un objeto de configuración para el componente “Ext.Toolbar”, esto es útil para cuando necesitamos utilizar la propiedad “defaults” que hemos estudiado en el tema de los paneles, además podemos utilizar todas las propiedades del componente (Toolbar). Ahora voy a mostrar como la propiedad “tbar” acepta una instancia del componente Toolbar:

```

1. //creates the tool bar
2. var toolbar = new Ext.Toolbar({
3. items: [
4. {text:'Back'},
5. {text:'Forward'},
6. {text:'Reload'},
7. {text:'Stop'},
8. {text:'Home'}
9.]
10. });
11.
12. this.win = new Ext.Window({
13. title: 'Quizzpot Explorer',
14. width: 600,
15. height:450,
16. tbar: toolbar, // <--- Toolbar
17. maximizable: true,
18. maskDisabled: true,
19. bodyStyle: 'background-color:#fff',
20. html: '<iframe id="container" src="http://www.google.com" style="width:100%;height:100%;border:none"></iframe>'
21. });
22.
23. this.win.show();

```

Como se han dado cuenta, estas tres maneras de crear una barra de herramientas son muy convenientes para diferentes situaciones, vamos a utilizar la última para continuar con nuestro ejercicio.

### Agregar íconos a los botones

Para agregar un ícono a un botón necesitamos crear una clase CSS donde asignaremos el ícono que queremos usar como background, vamos a definir que no se repita y ponerle la propiedad “!important” para que se pueda mostrar correctamente:

```

1. .back{
2. background: url(icons/arrow_left.png) no-repeat !important;
3. }

```

Ahora usamos la propiedad “iconCls” del objeto de configuración para la nuestro “toolbar”.

```
1. var toolbar = new Ext.Toolbar({
2. items: [
3. {text:'Back',iconCls:'back'}, //<--- adding an icon to the button
4. {text:'Forward'},
5. {text:'Reload'},
6. {text:'Stop'},
7. {text:'Home'}
8.]
9. });
```



Botón con ícono

Vamos a agregar el resto de los iconos a los botones faltantes; las clases de CSS han sido definidas en el HTML.

```
1. var toolbar = new Ext.Toolbar({
2. items: [
3. {text:'Back',iconCls:'back'},
4. {text:'Forward',iconCls:'forward'},
5. {text:'Reload',iconCls:'reload'},
6. {text:'Stop',iconCls:'stop'},
7. {text:'Home',iconCls:'home'}
8.]
9. });
```



Botones con íconos

Por defecto la alineación del ícono es la parte izquierda del botón, esto lo podemos configurar mediante la propiedad "iconAlign" la cual acepta como valor "top", "right", "bottom" o "left". Voy a utilizar la propiedad "defaults" del toolbar para aplicar esta configuración a todos los botones.

```

1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top' // <--- we change the icon position
4. },
5. items: [
6. {text:'Back',iconCls:'back'},
7. {text:'Forward',iconCls:'forward'},
8. {text:'Reload',iconCls:'reload'},
9. {text:'Stop',iconCls:'stop'},
10. {text:'Home',iconCls:'home'}
11.]
12. });

```



Cambiando la alineación de los íconos

Al hacer este cambio podemos ver como ha cambiado la posición del ícono, ahora se encuentra en la parte superior del botón, pero notemos que el ícono está alineado a la izquierda, para corregir esto necesitamos centralizarlo desde la clase CSS donde definimos la imagen.

```
1. .back{
2. background: url(icons/arrow_left.png) center 0px no-repeat !important;
3. }
```

Con esto veremos como los iconos se han centrado en el botón.



Íconos centrados en el botón

### Botones alineados a la derecha

Cuando necesitemos que los botones estén alineados a la derecha del panel, únicamente debemos utilizar el componente "Ext.Toolbar.Fill" para que nos posicione los botones en el lugar correcto.

```
1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top'
4. },
5. items: [
6. {text:'Back',iconCls:'back'},
7. {text:'Forward',iconCls:'forward'},
8. {text:'Reload',iconCls:'reload'},
9. {text:'Stop',iconCls:'stop'},
10. {text:'Home',iconCls:'home'},
11. new Ext.Toolbar.Fill(), // <--- we fill the empty space
12. {text:'Bookmarks',iconCls:'book'}// now the following buttons are in the right side
13.]
14.});
```



Botones alineados a la derecha

También podemos utilizar el atajo “->” para el componente “Ext.Toolbar.Fill”, de esta manera escribiremos menos.

```

1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top'
4. },
5. items: [
6. {text:'Back',iconCls:'back'},
7. {text:'Forward',iconCls:'forward'},
8. {text:'Reload',iconCls:'reload'},
9. {text:'Stop',iconCls:'stop'},
10. {text:'Home',iconCls:'home'},
11. '->', // <--- shortcut for the Ext.Toolbar.Fill class
12. {text:'Bookmarks',iconCls:'book'}// now the following buttons are in the right side
13.]
14. });

```



La pantalla resulta igual a la anterior

## Una caja de texto en la barra de herramientas

Vamos agregar una caja de texto a la barra de herramientas para que el usuario pueda introducir una dirección y al darle clic en el botón buscar o presionar la tecla "enter" el iFrame muestre la página solicitada.

```
1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top'
4. },
5. items: [
6. {text:'Back',iconCls:'back'},
7. {text:'Forward',iconCls:'forward'},
8. {text:'Reload',iconCls:'reload'},
9. {text:'Stop',iconCls:'stop'},
10. {text:'Home',iconCls:'home'},
11. '-' // <--- add a vertical separator bar between toolbar items
12. {xtype:'textfield',id:'url',width:250,enableKeyEvents:true}, //<--- the textfield
13. {iconCls:'goto'},
14. '>',
15. {text:'Bookmarks',iconCls:'book'}
16.]
17.});
```



Separador vertical, caja de texto y botón sin texto

El código anterior tiene tres puntos importantes, primero se está agregando un separador vertical utilizando un guión, este es un atajo para la clase "Ext.Toolbar.Separator", el segundo punto importante es que se está modificando el "xtype" del botón por "textfield" (para este caso), de esta manera cambiamos el tipo de componente especificando en este caso usaremos una caja de texto en lugar del botón, el tercer punto importante es que estamos definiendo un botón que no tiene texto, únicamente tiene un ícono.

## Agrupando botones

La nueva versión de Ext JS (v 3.0) nos permite formar grupos de botones, esto es muy útil para separar por funcionalidad los botones semejantes o que se relacionan de una u otra manera, para este ejemplo vamos agrupar la caja de texto y el botón buscar.

```
1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top'
4. },
5. items: [
6. {text:'Back',iconCls:'back'},
```

```

7. {text:'Forward',iconCls:'forward'},
8. {text:'Reload',iconCls:'reload'},
9. {text:'Stop',iconCls:'stop'},
10. {text:'Home',iconCls:'home'},
11. '-,{',
12. xtype: 'buttongroup', // <--- grouping the buttons
13. items:[
14. {xtype:'textfield', id:'url', width:250, enableKeyEvents: true},
15. {iconCls:'goto'}
16.]
17. },
18. '->',
19. {text:'Bookmarks',iconCls:'book'}
20.],
21. });

```



Botones agrupados

También podríamos ponerle un título al grupo usando la propiedad "title" al componente "buttongroup", pero para este ejemplo no lo haremos, puedes probar si lo deseas y ver el resultado.

### “Split buttons” y menús

Vamos hacer que el botón “back” muestre un menú donde aparezcan los sitios que hemos visitado, simulando el historial de navegación.

```

1. var toolbar = new Ext.Toolbar({
2. defaults:{
3. iconAlign: 'top'
4. },
5. items: [
6. {
7. text:'Back',iconCls:'back',
8. split: true, // <--- split the button
9. menu:{ // <--- add a menu to the button
10. items: [
11. {text:'Yahoo!'}, // <--- This is an item for the menu
12. {text:'Quizzpot'},
13. {text:'Site point'}
14.]
15. }
16. },
17. {text:'Forward',iconCls:'forward'},
18. {text:'Reload',iconCls:'reload'},
19. {text:'Stop',iconCls:'stop'},

```

```

20. {text:'Home',iconCls:'home'},
21. '-',
22. xtype: 'buttongroup',
23. items:[
24. {xtype:'textfield', id:'url', width:250, enableKeyEvents:true},
25. {iconCls:'goto'}
26.]
27. },
28. '->',
29. {text:'Bookmarks',iconCls:'book'}]
30. });

```



El código anterior muestra como se ha agregado un menú al botón “back”, es importante mencionar que cada opción del menú puede tener otro submenu.

### Asignar acciones a los botones

Hasta ahora tenemos lista la GUI de nuestro explorador, pero no hace absolutamente nada, a continuación voy a mostrar como es que podemos asignarle acciones a los botones al momento de darles clic.

Voy a modificar el botón “Home” para que cuando el usuario de clic sobre el botón nos lleve hacia la página de inicio, en este caso “google.com”.

```

1. //... código removido por cuestiones de simplicidad
2. {
3. text:'Home', iconCls:'home',
4. handler: function(){
5. this.gotoUrl('http://www.google.com');
6. }.createDelegate(this)
7. },
8. //... código removido por cuestiones de simplicidad

```

La propiedad “handler” recibe una función donde podemos escribir todas las acciones que necesitamos realizar cuando el usuario de clic sobre el botón, el método “createDelegate” es muy importante para cambiar el scope de la función anónima, de esta manera la variable “this” hará referencia al objeto donde está el método “gotoUrl” (ToolbarTutorial), dentro de la función anónima estoy mandando a ejecutar una función que todavía no he definido, pero su propósito será cambiar el “src” del iFrame.

Vamos a definir la función responsable de mostrar la página solicitada, este método lo usaremos en varias ocasiones.

```

1. com.quizzpot.tutorial.ToolbarTutorial = {
2. init: function(){
3. //código removido por cuestiones de simplicidad
4. },
5. //esta función cambia la url del iFrame
6. gotoUrl: function(url){
7. if(!Ext.isEmpty(url)){
8. if(!/^http://.test(url)){
9. url = 'http://' +url;
10. }
11.
12. var iframe = Ext.get('container');
13. iframe.dom.src = url;
14. Ext.getCmp('url').setValue(url);
15. this.win.setTitle(url + ' - Quizzpot Explorer');
16. }
17. }
18. }

```

La función anterior primero verifica que el parámetro “url” tenga algo, luego se asegura de que comience con “http://” de no ser así se lo agrega, luego cambia la propiedad “src” del iFrame y le asigna la url a la caja de texto y al título de la ventana. Ahora voy a modificar el botón “Search” para que cuando el usuario de clic sobre él, tome lo que hay en la caja de texto y llame a la función “gotoUrl”.

```
1. {iconCls:'goto',handler: this.search.createDelegate(this)}
```

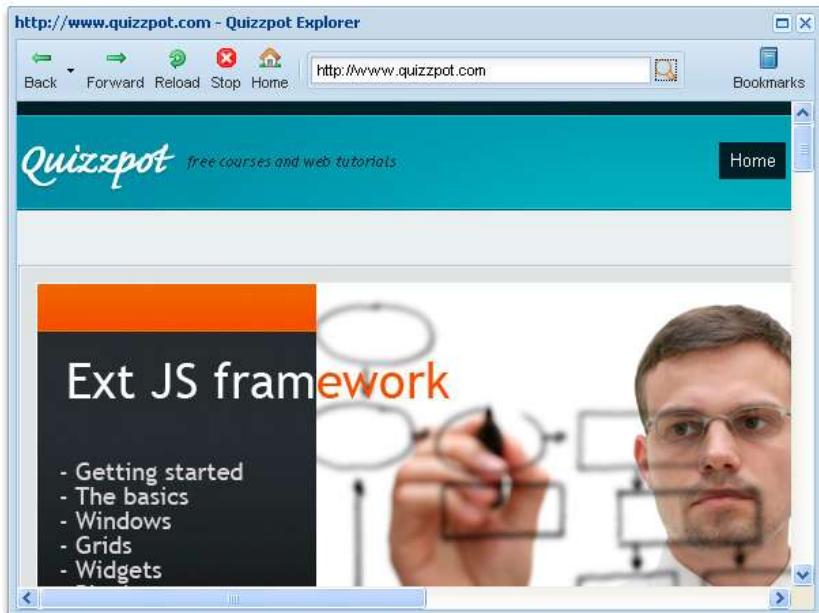
En el código anterior no creé una función anónima, sino que le asigné una función del objeto “ToolbarTutorial” la cual vamos a definir a continuación:

```

1. com.quizzpot.tutorial.ToolbarTutorial = {
2. init: function(){
3. //... no se muestra el código para simplificar las cosas
4. },
5.
6. search: function(btn,event){
7. this.gotoUrl(Ext.getCmp('url').getValue()); //<---
8. se toma lo que el usuario puso en la caja de texto
9. },
10. gotoUrl: function(url){
11. //... no se muestra el código para simplificar las cosas
12. }
13. }

```

La función “search” únicamente invoca a la función “gotoUrl” con lo que hay en la caja de texto.



Navegando en el explorador

## Conclusiones

En el tema de hoy vimos como crear una barra de herramientas, como usar los botones y como crear menús, además vimos como ponerles comportamiento a los botones, si hay alguna duda al respecto pueden realizarla en el foro o bien dejar sus comentarios en esta misma página.

## Barra de status en ventanas y paneles

Una barra de estado nos puede ayudar para informar al usuario lo que está sucediendo en la aplicación. Eliminaron este componente en la versión 3.0 de Ext JS, argumentando que era lo mismo que un Toolbar y que el componente no merecía ser parte del Framework. En este tutorial voy a mostrar como podemos recrear una barra de status con Ext 3.0.

### Material de apoyo

Antes de empezar es necesario descargar el material de apoyo, descomprimirlo y copiarlo al servidor Web que instalamos en el primer capítulo del curso.

El ejercicio que haremos será agregar una barra de estado a un ventana, en la parte derecha tendrá un reloj, además la ventana tendrá un área de texto donde el usuario podrá escribir y mientras lo hace en la barra de estado aparecerá el conteo de las palabras y cuando deje de escribir se mostrará un mensaje de "guardando", no vamos a guardar nada pues el objetivo es mostrar la forma de utilizar la barra de estado, puedes ver la demostración del ejercicio.

### Empaquetando el tutorial

Lo primero que tenemos que hacer es empaquetar el código que vamos a escribir en este tutorial.

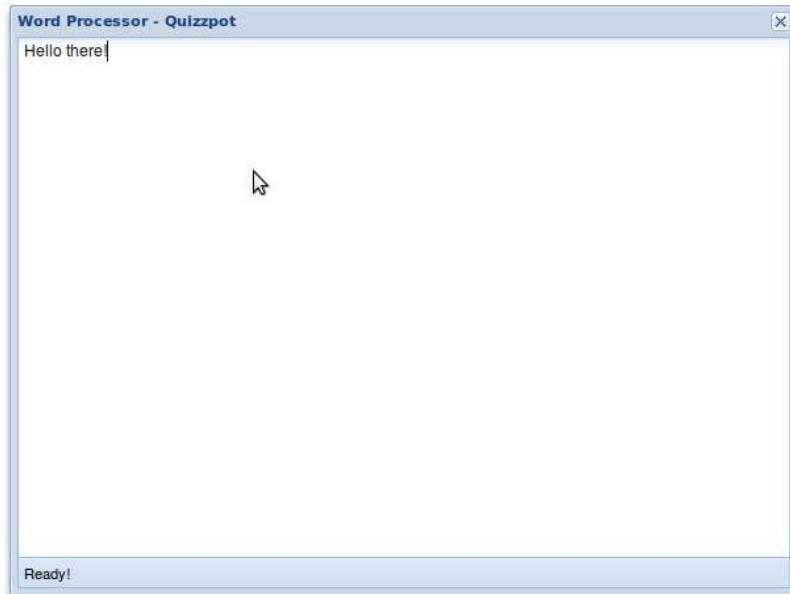
1. Ext.ns('com.quizzpot.tutorial');
- 2.
3. com.quizzpot.tutorial.StatusbarTutorial = {
4.     init: function(){
5.         //the code goes here
6.     }
7. }
- 8.
9. Ext.onReady(com.quizzpot.tutorial.StatusbarTutorial.init,com.quizzpot.tutorial.StatusbarTutorial);

## Crear la barra de estado

Para crear una barra de estado creamos una instancia del componente “Ext.Toolbar” y se la asignamos al componente Window mediante la propiedad “bbar”.

```
1. //creates the status bar
2. var statusbar = new Ext.Toolbar({
3. items:['Ready!']
4. });
5.
6. this.win = new Ext.Window({
7. title: 'Word Processor - Quizzpot',
8. width: 600,
9. height: 450,
10. bbar: statusbar, // <--- add the status bar to the window
11. bodyStyle:'background-color:#fff',
12. items: [
13. xtype: 'textarea',
14. id: 'word-textarea',
15. style: 'width:100%;height:100%;border:none;',
16. enableKeyEvents: true
17.]
18. });
19.
20. this.win.show();
```

El código anterior crea una barra de herramientas que solamente tiene un texto (Ready!), luego crea una ventana (la cual hereda de Ext.Panel) a la cual le asigna la barra de estado mediante la propiedad “bbar”, además la ventana cuenta con un “textarea” donde el usuario podrá escribir, más adelante vamos a ponerle algunos eventos a este último componente. Hasta ahora nos debe aparecer en la pantalla del explorador algo semejante a la siguiente imagen.



Una barra de estado sencilla

## Texto en el Statusbar

Lo que vamos a realizar en este paso es crear los contenedores de texto que vamos a estar usando.

```
1. //method vars
2. var status = new Ext.Toolbar.TextItem({id:'status',text:'Ready!'});
3. var clock = new Ext.Toolbar.TextItem({id:'clock',text: '00:00:00 AM'});
4. // instance var
```

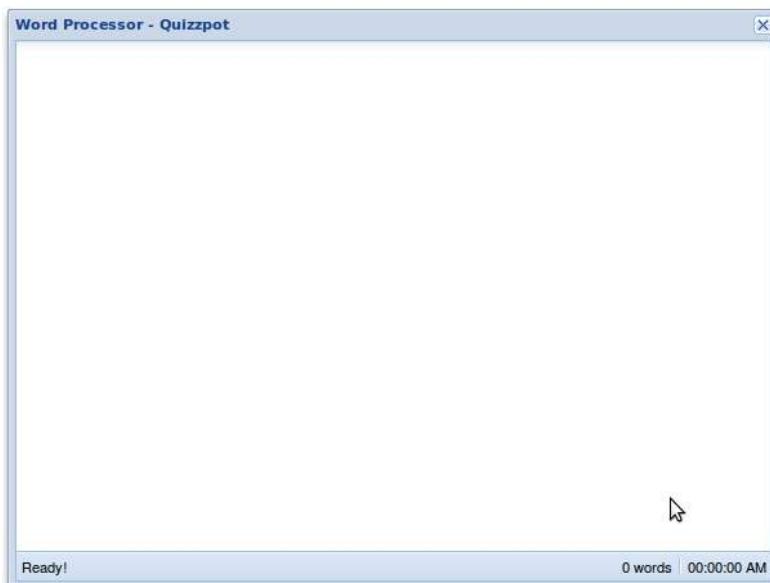
```

5. this.words = new Ext.Toolbar.TextItem({id:'words',text:'0 words'});
6.
7. var statusbar = new Ext.Toolbar({
8. items:[status,'->',this.words,'-',clock]
9. });

```

El código anterior crea tres “TextItems”, estos son contenedores de texto que podemos insertarle a una barra de herramientas; lo estoy haciendo de esta manera porque necesito asignarle un “id” a cada ítem para luego tener acceso a ellos y modificar su contenido, luego se los estoy asignando a la barra de estado.

Otro punto que es importante mencionar es que las variables “status” y “clock” están siendo creadas como variables del método mientras la variable “words” es una variable de instancia porque será utilizada en otro método del objeto “StatusbarTutorial” más adelante (todavía no he definido ese método). Hasta ahora debemos tener una pantalla semejante a esta:



Componente TextItem para desplegar texto en un Toolbar

### Programando un reloj

Vamos a programar el reloj que mostraremos en la parte de la derecha, así que necesitamos crear una tarea que se ejecute cada segundo y modifique el texto del “TextItem clock” que definimos en el paso anterior.

```

1. // Start a simple clock task that updates a div once per second
2. var updateClock = function(){
3. Ext.getCmp('clock').setText(new Date().format('g:i:s A'));
4. }
5.
6. //Configuration object for the task
7. var task = {
8. run: updateClock, //the function to run
9. interval: 1000 //every second
10. }
11.
12. //creates a new manager
13. var runner = new Ext.util.TaskRunner();
14. runner.start(task); //start running the task every one second

```

En el código anterior primeramente se está creando una función o rutina que será ejecutada continuamente, en esta solamente se está actualizando el texto del “clock” con la hora actual del sistema;

el segundo paso es crear un objeto de configuración para una tarea, donde es necesario definir dos parámetros, el “run” que es donde se define la función o rutina a ejecutar y el “interval” que es el tiempo en el cual se ejecutará continuamente, este tiempo se debe especificar en milisegundos; el tercer paso es crear una instancia del componente “TaskRunner” e iniciar la ejecución de las tareas deseadas, en este caso solamente estamos ejecutando una tarea, pero podemos ejecutar varias a la vez.



Un reloj en la barra de estado

### Contador de palabras

El siguiente paso será actualizar el contador de palabras cuando el usuario esté escribiendo, por lo tanto usaremos el evento “keypress” y “blur” para estar actualizando inmediatamente cuando el usuario ingresa una palabra nueva.

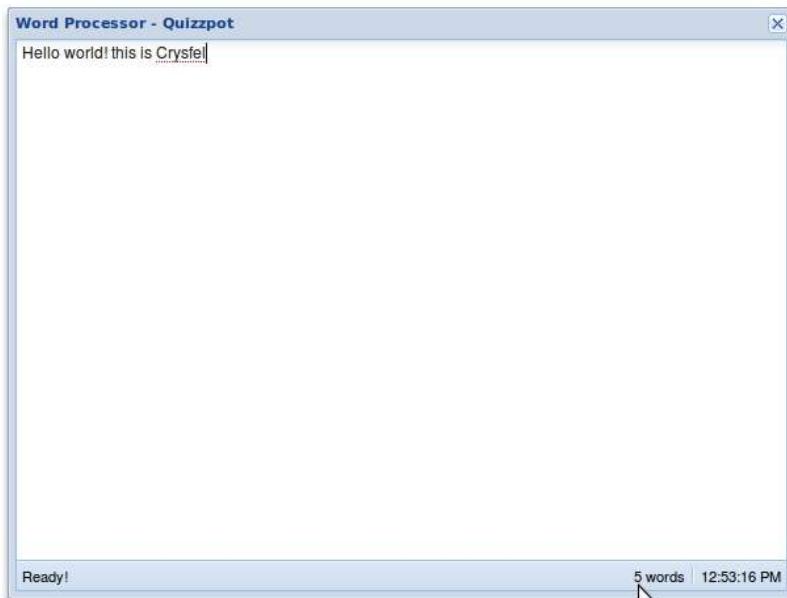
1. Ext.getCmp('word-textarea').on('keypress',this.countWords,this);
2. Ext.getCmp('word-textarea').on('blur',this.countWords,this);

Mediante el método “getCmp” podemos tomar el componente por su identificador (id), y utilizarlo normalmente, luego agregamos el “listener” a los eventos mencionados anteriormente.

Si has notado el segundo parámetro está haciendo referencia a un método de instancia el cual no hemos definido aún, y el tercer parámetro es el “scope” que le estamos asignando. Vamos a definir el método que se encargará de contar las palabras en el texto ingresado por el usuario inmediatamente después del método “init”.

1. , // <-- do not forget the comma between methods
- 2.
3. countWords: function(textarea){
4. var value = textarea.getValue(); //get the string
5. var wc = 0; // word counter
- 6.
7. if(!Ext.isEmpty(value)){ // if there is something in the textfield
8. wc = value.match(/\b/g); //find the spaces
9. wc = wc ? wc.length / 2 : 0; //count the words
10. }
- 11.
12. this.words.setText(wc + ' words'); //print the counter on the status bar
13. }

El código anterior toma el texto que ingreso el usuario, luego inicializa un contador para las palabras en cero, revisa si se ha introducido algo y cuenta las palabras buscando espacios y calculando el total de las palabras, por último actualiza la barra de estado con el nuevo valor, la pantalla nos debe quedar algo semejante a esto:



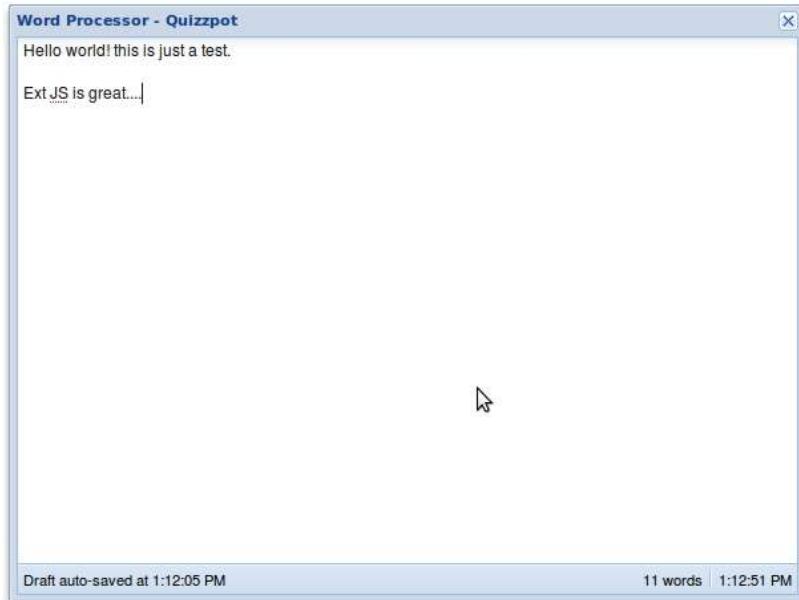
Contador de palabras

### Auto guardado

Vamos a simular un “autosave” de lo que el usuario está escribiendo, esto lo haremos mostrando un mensaje de “Saving draft...” en la barra de estado y después de un segundo mostraremos la hora de la última vez que se guardó, no vamos a guardar en alguna base de datos por cuestiones de simplicidad, pero hacer esta implementación es muy sencilla ya que simplemente necesitas enviar el contenido de la caja de texto al servidor utilizando Ajax, cuando el servidor responda le avisas al usuario.

```
1. Ext.getCmp('word-textarea').on('keypress', function(textarea){
2. var text = Ext.getCmp('status');
3. text.setText('Saving draft...');
4. (function(){
5. text.setText('Draft auto-saved at ' + new Date().format('g:i:s A'));
6. }).defer(2000);
7. }, this, {buffer:3000});
```

Tenemos que escribir el código anterior dentro de la función “init” junto con los otros “listeners” que agregamos anteriormente; aquí quiero destacar que la función “on” acepta un objeto de configuración en el cuarto parámetro donde le estamos indicando la propiedad “buffer” la cual permite hacer un “delay” al ejecutar el “listener” de 3000 milisegundos; en otras palabras esta función se ejecutará cuando se presiona una tecla y pasen tres segundos, con esto evitamos que se ejecute por cada letra que el usuario introduce logrando guardar la información cuando hay una inactividad de tres segundos. Otro punto importante del código anterior es que estamos usando una función anónima la cual se ejecuta después de dos segundos, podía haber utilizado un “setTimeout”, pero quería mostrarles como utilizar el método “defer”.



Autoguardado del texto

### Conclusiones

Aunque en la versión de Ext 3.0 eliminaron la barra de estado, aún podemos recrearla, precisamente por eso la borraron porque no tenía mucho sentido tener un componente que era prácticamente igual al "Toolbar", por otro lado la comunidad de Ext ha creado un plugin para el "Statusbar" el cual trae algunos métodos para facilitar la tarea.

## Las Pestañas o Tabs

Las pestañas son muy útiles para dividir la información por secciones o por categorías, en Ext JS es muy sencillo realizar esto, este tutorial se muestra como hacerlo.

### Material de apoyo

Antes de continuar asegúrate de descargar los recursos para este tema, descomprimir el ZIP y copiar el contenido dentro de la carpeta "paneles" que creamos al inicio de este capítulo.

Para este tutorial vamos a crear una ventana, dentro de ésta vamos a colocar un TabPanel con varios tabs que crearemos dinámicamente en tiempo de ejecución, te invito que veas el demo que he preparado.



Ejemplo final

### El TabPanel

Un TabPanel es un contenedor de pestañas o tabs, no existe un componente "Tab" pero podemos agregarle el componente "Ext.Panel", "Ext.grid.GridPanel", "Ext.tree.TreePanel" y cualquier componente que herede del "Panel".

A continuación vamos a crear el TabPanel de la siguiente manera:

```

1. //Creamos el primer tab
2. var home = new Ext.Panel({
3. title:'Home',
4. iconCls: 'home-icon',
5. html: 'This is the home page example'
6. });
7.
8. this.tabs = new Ext.TabPanel({
9. items: home //le agregamos el primer tab
10. });

```

En el código anterior primero se creó un panel que será el home, luego se ha creado el “TabPanel” y únicamente se le está asignando la variable “home” para convertirse en el primer tab, hasta ahora no se ve nada en la pantalla, esto es porque no lo hemos “renderizado”, para mostrarlo podemos utilizar la propiedad “renderTo”, o el método “render” o en este caso utilizar una ventana que contenga el “TabPanel” de la siguiente manera:

```

1. var win = new Ext.Window({
2. title:'Tabs example',
3. width:600,
4. height:500,
5. bodyStyle: 'background-color:#fff;',
6. items: this.tabs //le asignamos el tabpanel
7. });
8. win.show();

```

El código anterior debe ser familiar para nosotros, pues ya lo hemos estudiado en temas anteriores, si actualizamos la página donde se ejecuta este script veremos algo semejante a la siguiente imagen.



Tab panel en ventana con solo un tab

Si eres observador notarás que el tab “home” no aparece activado hasta que le damos clic con el Mouse y aparece su contenido; si queremos que el contenido aparezca desde el principio tenemos que utilizar la propiedad “activeTab” y asignarle el índice del tab que queremos activar, entonces el código del “TabPanel” quedaría de la siguiente manera:

```

1. this.tabs = new Ext.TabPanel({
2. border: false,
3. activeTab: 0, //-->activar el primer tab
4. items:[home]
5. });

```

Es importante notar que los índices comienzan en “0”, por lo tanto para activar el primero tenemos que activar el tab con el índice cero.



Primer tab activado automáticamente

## Agregar Tabs en tiempo de ejecución

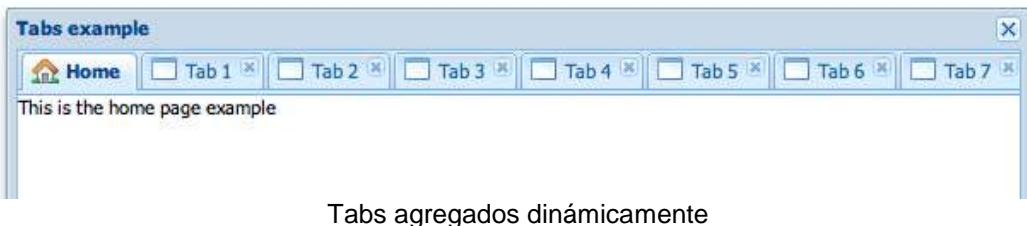
Ahora vamos a agregar varios tabs en tiempo de ejecución, es decir dinámicamente, para eso vamos a escribir el código necesario dentro del método “addTab” que se encuentra en el objeto “TabPanelTutorial” el cual ya viene definido en el material de apoyo que descargamos, dentro de este método vamos a crear un panel y se lo vamos a agregar el “TabPanel” que hemos creado anteriormente.

```
1. addTab: function(i){
2. //aquí va el código para agregar un nuevo tab
3. var tab = new Ext.Panel({
4. title: 'Tab '+i,
5. closable: true, //<-- este tab se puede cerrar
6. iconCls: 'app-icon',
7. tbar:[{iconCls:'save-icon'}, {iconCls:'spell-icon'}, {iconCls:'search-icon'}, {iconCls:'send-icon'}, {iconCls:'print-icon'}],
8. html: 'This is the content for the tab number '+i
9. });
10.
11. this.tabs.add(tab); //con esto le agregamos el tab
12. }
```

El código anterior crea un panel con la propiedad “closable”, esto hace que el tab pueda ser cerrado por el usuario en cualquier momento, además se le agregó una barra de herramientas, la cual no hace nada pero quiero enfatizar que podemos utilizar cualquier configuración del componente Panel.

Por último necesitamos invocar el método que acabamos de crear mediante un ciclo dentro del método “init” de la siguiente manera:

```
1. init: function(){
2. //... código removido
3. win.show();
4.
5. //Creamos 10 tabs
6. for(var i=0;i<10;i++){
7. this.addTab(i+1);
8. }
9. },
```



Tabs agregados dinámicamente

## Agregar un scroll a los tabs

Hasta ahora tenemos un problema, y es que no aparecen los últimos tabs porque la ventana es más pequeña, esto lo podemos solucionar agregando un “scroll” para que podamos desplazar los tabs y mirar todos por medio de la propiedad “enableTabScroll” del “TabPanel” de la siguiente manera:

```
1. this.tabs = new Ext.TabPanel({
2. border: false,
3. activeTab: 0,
4. enableTabScroll:true, //<-- muestra un scroll para los tabs
5. items:[home]
6. });
```



TabPanel con scroll para mover los tabs

El código completo para este tutorial queda de la siguiente manera:

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.TabPanelTutorial = {
4. init: function(){
5. //Aquí está el código inicial
6. var home = new Ext.Panel({
7. title:'Home',
8. iconCls: 'home-icon',
9. html: 'This is the home page example'
10. });
11.
12. this.tabs = new Ext.TabPanel({
13. border: false,
14. activeTab: 0,
15. enableTabScroll:true,
16. items:[home]
17. });
18.
19. var win = new Ext.Window({
20. title:'Tabs example',
21. width:600,
22. height:500,
23. bodyStyle: 'background-color:#fff;',
24. items: this.tabs
25. });
26. win.show();
27.
28. for(var i=0;i<10;i++){
29. this.addTab(i+1);
30. }
31. },
32.
33. addTab: function(i){
34. //here the code to add a new tab
35. var tab = new Ext.Panel({
36. title: 'Tab '+i,
37. closable: true, //<-- this tab is closable
38. iconCls: 'app-icon',
39. tbar:[{iconCls:'save-icon'}, {iconCls:'spell-icon'}, {iconCls:'search-icon'}, {iconCls:'send-icon'}, {iconCls:'print-icon'}],
40. html: 'This is the content for the tab number '+i
41. });
42.
43. this.tabs.add(tab);
44. }
45. }
46.
47. Ext.onReady(com.quizzpot.tutorial.TabPanelTutorial.init,com.quizzpot.tutorial.TabPanelTutorial);
```

## Conclusiones

El TabPanel es un Widget muy utilizado para crear interfaces usables, además es muy flexible y fácil de utilizar, te recomiendo mirar el API para que conozcas algunas otras propiedades de este componente y como siempre si tienes dudas o sugerencias favor de hacerlas en los comentarios o bien suscríbete en el foro el cual esta creciendo poco a poco con la ayuda de todos ustedes.

## Los Formularios

Se muestra como crear formularios, validaciones, diferentes controles para capturar información e interactuar con un servidor para obtener y guardar información.

## Formularios y campos comunes en Ext JS

Entre los múltiples tipos de paneles que nos ofrece Ext JS, tenemos el componente Ext.FormPanel, un componente que nos proporciona toda la funcionalidad de un formulario común en HTML pero con métodos y funcionalidades propias de ExtJS. El componente Ext.FormPanel tiene un tipo de diseño (“layout”) por defecto tipo “form”, este tipo de diseño alinea de una manera inteligente cada uno de los componentes (etiqueta-componente) del formulario. La característica única de los Ext.FormPanel es que ya tiene implementado el guardado y cargado de datos de una manera segura y totalmente configurable.

### Material de apoyo

Ahora descargamos el material de apoyo, descomprimimos su contenido, creamos una carpeta que se llame “formularios” dentro de la carpeta “curso” en la cual hemos estado trabajando, luego copiamos los archivos que descomprimimos anteriormente. Recuerden que este código lo vamos a ocupar durante todo el tema de “Formularios” así que en los tutoriales siguientes agregaremos líneas a este código para añadirle funcionalidad a nuestro formulario además de que estamos utilizando la versión 3.0 del Framework de Ext JS.

El objetivo es que al final del tema “Formularios”, tengamos un formulario funcionando correctamente con todas las funcionalidades y validaciones correspondientes.

### Empaquetando el tutorial

Vamos a empaquetar el código para evitar conflictos con otras variables.

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. Ext.BLANK_IMAGE_URL = '../ext-3.0/resources/images/default/s.gif';
4.
5. com.quizzpot.tutorial.FormTutorial = {
6. init: function(){
7. //AQUÍ va el código del tutorial
8. }
9.
10. }
11.
12. Ext.onReady(com.quizzpot.tutorial.FormTutorial.init,com.quizzpot.tutorial.FormTutorial);
```

### Formulario

A continuación vamos a construir un formulario con los campos más utilizados; dos campos de texto, un grupo de checkbox, un campo oculto, un grupo de radiobuttons y dos botones. Primero necesitamos crear un formulario donde se alojarán los campos para que el usuario capture la información necesaria, esto lo hacemos de la siguiente manera:

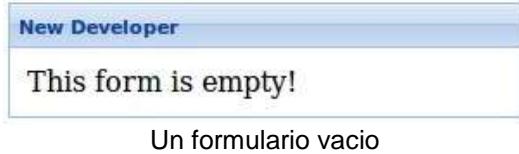
```
1. //Creamos un formulario
2. this.form= new Ext.FormPanel({
3. title:'New Developer',
4. renderTo: 'frame',
5. defaults:{xtype:'textfield'}, //componente por defecto del formulario
6. bodyStyle:'padding: 10px', //alejamos los componentes del formulario de los bordes
```

```
7. html: 'This form is empty!' //<-- en el siguiente paso vamos a quitar esta propiedad
8. });


```

Los formularios heredan las propiedades y métodos del componente "*Ext.Panel*" por lo tanto la configuración que hemos realizado ya es conocida por nosotros, si no es así te invito a darle un repaso al tema de los paneles.

El código anterior genera la siguiente pantalla:



Hasta este punto no hay diferencia visual entre un formulario y un panel, esto es porque aún no le hemos agregado ningún campo, vamos a ver detalladamente los componentes más comunes a continuación.

### Campos de texto

Como hemos visto, en temas anteriores, podemos crear los componentes utilizando la palabra reservada "new" seguido del componente que necesitamos instanciar, o bien podemos crearlos mediante objetos de configuración utilizando la propiedad "xtype" para distinguir entre los componentes disponibles. El componente que nos permite crear cajas de texto es el "Ext.form.TextField" y su "xtype" es "textfield", por ejemplo:

```
1. //Creamos una instancia del textfield
2. var name = new Ext.form.TextField({
3. fieldLabel:'Name',
4. name:'txt-name',
5. emptyText:'Your name...',
6. id:"id-name"
7. });
8.
9. //Creamos un formulario
10. this.form= new Ext.FormPanel({
11. title:'New Developer',
12. renderTo: 'frame',
13. defaults:{xtype:'textfield'}, //componente por defecto del formulario
14. bodyStyle:'padding: 10px', //alejamos los componentes del formulario de los bordes
15. items:[
16. name, // le asignamos la instancia que creamos anteriormente
17. {
18. fieldLabel:'Email', // creamos un campo
19. name:'txt-email', // a partir de una
20. value:'default@quizzpot.com<script type="text/javascript">
21. /* <! [CDATA[*/
22. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2)c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
;l.parentNode.replaceChild(s,l);}catch(e){}})();
23. /*]]> */
24. </script>', //configuración
25. id:"id-email"
26. }
27.]
28. });


```

En el código anterior se han creado dos campos de texto de dos maneras distintas, una utilizando una instancia del componente *TextField* y la otra usando un objeto de configuración, cada desarrollador puede elegir la opción que más le convenga dependiendo las circunstancias.

Campos de texto

A continuación voy a mencionar las propiedades que utilizamos:

- "*fieldLabel*": esta propiedad define el texto que acompaña a cada componente del formulario.
- "*emptyText*": esta propiedad define el texto que contendrá el campo cuando se encuentra vacío.
- "*name*": es el nombre con el cual se envían los datos que contiene el campo al servidor, es exactamente igual al "*name*" que estamos acostumbrados a usar en formularios comunes.
- "*value*": es el valor por defecto con que aparece en el componente, útil para cuando queremos editar un registro y mostrar la información que actualmente está capturada.

### **Checkbox**

Los checkboxes son utilizados para seleccionar uno o varios items de una lista, o simplemente para activar o desactivar alguna bandera o permiso en un sistema, para este ejemplo voy a poner un campo que se llame "activo" utilizando un objeto de configuración, también podemos crear una instancia utilizando el componente "*Ext.form.Checkbox*":

```

1. // código removido por cuestiones de simplicidad...
2.
3. // creamos un formulario
4. this.form= new Ext.FormPanel({
5. title:'New Developer',
6. renderTo: 'frame',
7. defaults:{xtype:'textfield'},
8. bodyStyle:'padding: 10px',
9. items:[
10. name,
11. {
12. fieldLabel:'Email',
13. name:'txt-email',
14. value:'default@quizzpot.com<script type="text/javascript">
15. /* <![CDATA[*/
16. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})();}
17. /*]]> */
18. </script>',
19. id:"id-email"
20. },{
21. xtype: 'checkbox', //definimos el tipo de componente
22. fieldLabel: 'Active',// le asignamos un label
23. name: 'chk-active', //y un "name" para que lo recojamos en el servidor...
24. id: 'id-active'// ...cuando el formulario sea enviado
25. }
26.]
27. });

```

Por otro lado cuando queremos agrupar varios checkboxes necesitamos utilizar el componente "*Ext.form.CheckboxGroup*" el cual permite manipular de una forma sencilla cualquier cantidad de checkboxes.

1. // creamos un grupo de checkboxes

```

2. var checkboxes = new Ext.form.CheckboxGroup({
3. fieldLabel:'Interests',
4. columns:2,//mostrar dos columnas de checkboxes
5. items:[
6. {boxLabel: 'JavaScript', name: 'cb-js', checked: true}, //campo marcado desde el principio
7. {boxLabel: 'HTML', name: 'cb-html'},
8. {boxLabel: 'CSS', name: 'cb-css'},
9. {boxLabel: 'Otros', name: 'cb-otros'}
10.]
11. });
12.
13. //Creamos un formulario
14. this.form= new Ext.FormPanel({
15. title:'New Developer',
16. renderTo: 'frame',
17. defaults:{xtype:'textfield'},
18. bodyStyle:'padding: 10px',
19. items:[
20. name,
21. {
22. fieldLabel:'Email',
23. name:'txt-email',
24. value:'default@quizzpot.com<script type="text/javascript">
25. /* <![CDATA[*/
26. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})();}
27. /*]]> */
28. </script>',
29. id:"id-email"
30. },{
31. xtype: 'checkbox', //definimos el tipo de componente
32. fieldLabel: 'Active',// le asignamos un label
33. name: 'chk-active',//y un "name" para que lo recojamos en el servidor...
34. id: 'id-active'// ...cuando el formulario sea enviado
35. },
36. checkboxes //<-- grupo de checkboxes
37.]
38. });

```

The screenshot shows a web-based form titled "New Developer". The form includes a text input for "Name" with placeholder text "Your name...", a text input for "Email" with the value "default@quizzpot.com", and a checkbox labeled "Active" which is checked. Below these are four checkboxes under the heading "Interests": "JavaScript" (checked), "HTML" (unchecked), "CSS" (unchecked), and "Otros" (unchecked).

Checkboxes con Ext JS

Los grupos de checkboxes aceptan la propiedad “*items*” la cual debe contener un arreglo de objetos de configuración para checkbox o instancias del checkbox.

## Radiobuttons

Los radiobutton son usados para seleccionar una sola opción de varios items, este componente se crea muy semejante a los checkboxes, solo que usamos el componente “*Ext.form.RadioGroup*” para agrupar varios radios.

```
1. //código removido por cuestiones de simplicidad
2.
3. //Creamos un grupo de opciones con radiobuttons
4. var radios = new Ext.form.RadioGroup({
5. fieldLabel: 'Favorite Framework',
6. columns: 2, //muestra los radiobuttons en dos columnas
7. items: [
8. {boxLabel: 'Ext Js', name: 'framework', inputValue: 'Ext js', checked: true},
9. {boxLabel: 'Dojo', name: 'framework', inputValue: 'Dojo'},
10. {boxLabel: 'Mootools', name: 'framework', inputValue: 'Mootools'},
11. {boxLabel: 'jQuery', name: 'framework', inputValue: 'jQuery'},
12. {boxLabel: 'prototype', name: 'framework', inputValue: 'prototype'},
13. {boxLabel: 'YUI', name: 'framework', inputValue: 'yui'}
14.]
15. });
16.
17. //Creamos un formulario
18. this.form= new Ext.FormPanel({
19. title:'New Developer',
20. renderTo: 'frame',
21. defaults:{xtype:'textfield'},
22. bodyStyle:'padding: 10px',
23. items:[
24. name,
25. {
26. fieldLabel:'Email',
27. name:'txt-email',
28. value:'default@quizzpot.com<script type="text/javascript">
29. /* <![CDATA[*/
30. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})();}
31. /*]]> */
32. </script>',
33. id:"id-email"
34. },
35. xtype: 'checkbox',
36. fieldLabel: 'Active',
37. name: 'chk-active',
38. id: 'id-active'
39.],
40. checkboxes,
41. radios // <-- grupo de radios
42.]
43. });
```

**New Developer**

|                     |                                                                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name:               | <input type="text" value="Your name..."/>                                                                                                                                                      |
| Email:              | <input type="text" value="default@quizzpot.com"/>                                                                                                                                              |
| Active:             | <input type="checkbox"/>                                                                                                                                                                       |
| Interests:          | <input checked="" type="checkbox"/> JavaScript <input type="checkbox"/> HTML<br><input type="checkbox"/> CSS <input type="checkbox"/> Otros                                                    |
| Favorite Framework: | <input checked="" type="radio"/> Ext Js <input type="radio"/> Dojo<br><input type="radio"/> Mootools <input type="radio"/> jQuery<br><input type="radio"/> prototype <input type="radio"/> YIU |

Grupo de Radiobuttons

Normalmente los radios son usados en conjunto, pero si por alguna extraña razón necesitas solamente uno o en lugar de usar objetos de configuración en la propiedad “items” del RadioGroup puedes utilizar el componente “Ext.form.Radio”.

### Campos ocultos

Los campos ocultos nos sirven para enviar información al servidor que el usuario no le interesa saber, por ejemplo algún “id” del registro que se esta editando, o algún token de seguridad, etc. Ext JS cuenta con el componente “Ext.form.Hidden” el cual nos permite lograr esta funcionalidad.

```

1. //código removido por cuestiones de simplicidad
2.
3. //Creamos un formulario
4. this.form= new Ext.FormPanel({
5. title:'New Developer',
6. renderTo: 'frame',
7. defaults:{xtype:'textfield'},
8. bodyStyle:'padding: 10px',
9. items:[
10. name,
11. {
12. fieldLabel:'Email',
13. name:'txt-email',
14. value:'default@quizzpot.com<script type="text/javascript">
15. /* <![CDATA[*/
16. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){{}();}
17. /*]]> */
18. </script>',
19. id:"id-email"
20. },{
21. xtype: 'checkbox',
22. fieldLabel: 'Active',
23. name: 'chk-active',
24. id: 'id-active'
25. },
26. checkboxes,
27. radios,
28. {
29. xtype:'hidden',//<-- campo oculto (hidden)
30. name:'h-type', //el nombre con que se envia al servidor

```

```

31. value:'developer'//el valor que contendrá
32. }
33.]
34. });

```

Es de suma importancia definir la propiedad "name" así como la propiedad "value" para asignarle el contenido a la variable que será enviada al servidor, adicionalmente podemos asignarle un "id" para que podamos modificar el valor del campo de una manera sencilla más adelante.

### **Botones en el formulario**

Podemos asignarle botones al formulario para que al ser presionados realicen las acciones correspondientes, por ahora únicamente voy a crear los botones sin ninguna acción.

```

1. //Creamos un formulario
2. this.form= new Ext.FormPanel({
3. title:'New Developer',
4. renderTo: 'frame',
5. defaults:{xtype:'textfield'},
6. bodyStyle:'padding: 10px',
7. items:[
8. name,
9. {
10. fieldLabel:'Email',
11. name:'txt-email',
12. value:'default@quizzpot.com<script type="text/javascript">
13. /* <![CDATA[*/
14. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l));}catch(e){}})();}
15. /*]]> */
16. </script>',
17. id:"id-email"
18. },
19. xtype: 'checkbox',
20. fieldLabel: 'Active',
21. name: 'chk-active',
22. id: 'id-active'
23. },
24. checkboxes,
25. radios,
26. {
27. xtype:'hidden',
28. name:'h-type',
29. value:'developer'
30. }
31.],
32. buttons:[{text:'Save'}, {text:'Cancel'}] //--- botones del formulario
33. });

```

The screenshot shows a 'New Developer' form dialog box. It includes fields for Name, Email, Active status, and various interests like JavaScript, HTML, CSS, and others. It also has sections for favorite frameworks like Ext Js, Dojo, Mootools, jQuery, prototype, and YIU. At the bottom are 'Save' and 'Cancel' buttons.

Botones en un formulario

Podemos alinear la posición de los botones a la derecha, izquierda o al centro, por defecto están alineados al centro, pero mediante la propiedad “buttonAlign” podemos definir el lugar donde aparecerán los botones, para alinearlos a la derecha haríamos los siguientes:

```

1. //creamos un formulario
2. this.form= new Ext.FormPanel({
3. title:'New Developer',
4. renderTo: 'frame',
5. defaults:{xtype:'textfield'},
6. bodyStyle:'padding: 10px',
7. items:[
8. name,
9. {
10. fieldLabel:'Email',
11. name:'txt-email',
12. value:'default@quizzpot.com<script type="text/javascript">
13. /* <![CDATA[*/
14. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
;l.parentNode.replaceChild(s,l);}catch(e){{}()}());
15. /*]]> */
16. </script>',
17. id:"id-email"
18. },
19. xtype: 'checkbox',
20. fieldLabel: 'Active',
21. name: 'chk-active',
22. id: 'id-active'
23. },
24. checkboxes,
25. radios,
26. {
27. xtype:'hidden',
28. name:'h-type',
29. value:'developer'
30. }
31.],
32. buttonAlign: 'right', //<--botones alineados a la derecha
33. buttons:[{text:'Save'}, {text:'Cancel'}] //botones del formulario
34. });

```

**New Developer**

|                                                                           |                                                                                                                                                                                                |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Name:                                                                     | Your name...                                                                                                                                                                                   |
| Email:                                                                    | default@quizzpot.com                                                                                                                                                                           |
| Active:                                                                   | <input type="checkbox"/>                                                                                                                                                                       |
| Interests:                                                                | <input checked="" type="checkbox"/> JavaScript <input type="checkbox"/> HTML<br><input type="checkbox"/> CSS <input type="checkbox"/> Otros                                                    |
| Favorite Framework:                                                       | <input checked="" type="radio"/> Ext Js <input type="radio"/> Dojo<br><input type="radio"/> Mootools <input type="radio"/> jQuery<br><input type="radio"/> prototype <input type="radio"/> YIU |
| <input type="button" value="Save"/> <input type="button" value="Cancel"/> |                                                                                                                                                                                                |

Botones alineados a la derecha

### Ventana que alojara el formulario

Solo por cuestiones visuales, vamos a ocupar una ventana para alojar nuestro formulario ahí, así que es necesario quitar la propiedad “`renderTo: 'frame'`” e insertar el formulario dentro de la ventana que crearemos, también vamos a mover los botones a la ventana, el título y el estilo del body de la siguiente manera:

```

1. //Creamos un formulario
2. this.form= new Ext.FormPanel({
3. border:false, // <-- Le quitamos el borde al formulario
4. defaults:{xtype:'textfield'}, //componente por default del formulario
5. items:[
6. name, // le asignamos la instancia que creamos anteriormente
7. {
8. fieldLabel:'Email', // creamos un campo
9. name:'txt-email', // a partir de una
10. value:'default@quizzpot.com<script type="text/javascript">
11. /* <![CDATA[*/
12. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
;l.parentNode.replaceChild(s,l);}catch(e){}})();
13. /*]]> */
14. </script>', //configuración
15. id:"id-email"
16. },
17. xtype: 'checkbox', //definimos el tipo de componente
18. fieldLabel: 'Active',// le asignamos un label
19. name: 'chk-active',//y un "name" para que lo recojamos en el servidor...
20. id: 'id-active'// ...cuando el formulario sea enviado
21. },
22. checkboxes, //grupo de checkboxes
23. radios, // grupo de radios
24. {
25. xtype:'hidden',//campo oculto (hidden)
26. name:'h-type', //el nombre con que se envia al servidor
27. value:'developer'//el valor que contendrá
28. }
29.]
30. });

```

```

31.
32. //Creamos la ventana que contendrá el formulario
33. var win = new Ext.Window({
34. title: 'New Developer',
35. width:300,
36. height:300,
37. bodyStyle:'background-color:#fff;padding: 10px',
38. items:this.form, //Le asignamos el formulario solamente
39. buttonAlign: 'right', //botones alineados a la derecha
40. buttons:[{text:'Save'},{text:'Cancel'}] //botones del formulario
41. });
42.
43. win.show();

```



Ventana con un formulario

### Conclusiones

El día de hoy hemos visto como crear un formulario de manera muy fácil y sencilla, el código completo lo puedes descargar en la parte superior derecha de esta página, en el próximo tutorial veremos cómo añadirle un combo box con datos cargados de manera local y remota a nuestro formulario, y también le agregaremos algunos campos más que nos proporciona Ext JS.

### Combo box cargados local y remotamente (Ajax)

El día de hoy vamos a hablar de un componente que es muy completo: el ComboBox. Veremos cómo configurar un Combo Box de manera local y de manera remota. Describiremos algunas de sus propiedades en la configuración, crearemos un plantilla para que nuestro Combo Box tenga un formato agradable, y al final daremos un recorrido por las diferentes variaciones del Combo Box que nos ofrece ExtJS, por ejemplo el TimeField.

### Material de apoyo

Para poder continuar necesitamos descargar el material de apoyo. Recuerda que este tutorial pertenece a la parte de formularios, y al final del tutorial agregaremos nuestro ComboBox al formulario pero para evitar algunas confusiones haremos este tutorial en un archivo diferente.

### Empaquetando el tutorial.

Procedemos a empaquetar nuestro código.

```

1. Ext.ns('com.quizzpot.tutorial');
2.
3. Ext.BLANK_IMAGE_URL = './ext-3.0/resources/images/default/s.gif';
4.
5. com.quizzpot.tutorial.ComboBoxTutorial= {
6. init: function(){
7. //aqui va el codigo del tutorial
8. }
9. }
10. Ext.onReady(com.quizzpot.tutorial.ComboBoxTutorial.init,com.quizzpot.tutorial.ComboBoxTutorial);

```

### Ventana

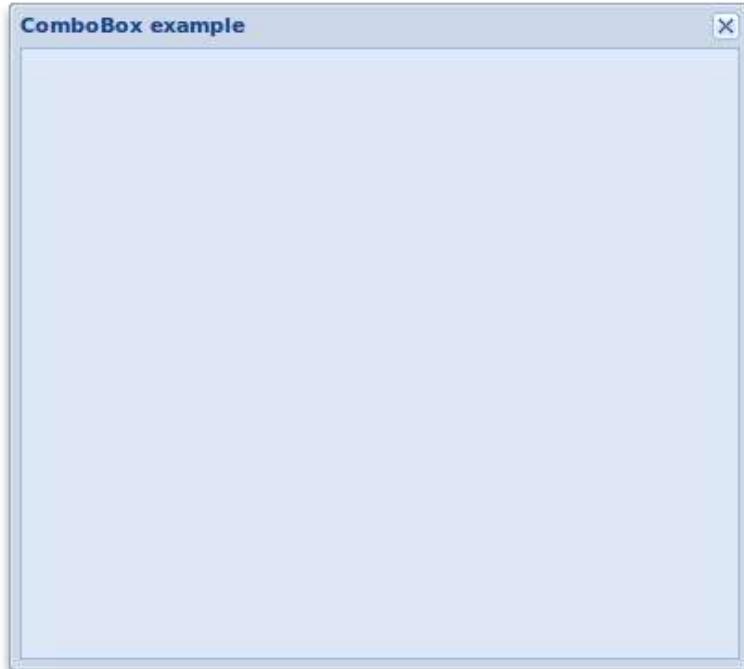
A continuación vamos a crear una ventana que alojará nuestros diferentes tipos de ComboBox. Esta ventana nos sirve para visualizar nuestros distintos ComboBoxes que crearemos en este tutorial.

```

1. var win=new Ext.Window({
2. title: 'ComboBox example',
3. bodyStyle:'padding: 10px', //alejamos los componentes de los bordes
4. width:400,
5. height:360,
6. layout:'form' //tipo de organización de los componentes
7. });
8. win.show();

```

Del código anterior podemos resaltar la propiedad “*layout:’form’*”, que para una ventana, el layout por defecto es el “*fit*”, por lo tanto sobre escribimos la propiedad para mostrar los componentes distribuidos como un formulario.



Ventana que contendrá los ComboBox

### ComboBox Local

Una vez creada nuestra ventana ahora vamos a crear nuestros ComboBoxes; lo primero que necesitamos son los datos que se cargarán en nuestro ComboBox, la manera más simple de cargar datos en nuestro ComboBox es mediante el uso de un arreglo de datos.

```

1. var data=['Code Igniter','Cake Php','Symfony','Zend'];
2.
3. var comboLocal =new Ext.form.ComboBox({
4. fieldLabel:'Frameworks PHP',
5. name:'cmb-data',
6. forceSelection:true,
7. store:data,
8. emptyText:'Select a framework...',
9. triggerAction: 'all',
10. //hideTrigger:true,
11. editable:false,
12. //minChars:3
13. });

```

El código anterior genera un ComboBox, con datos correspondientes a algunos Frameworks existentes de PHP en el mercado. Vamos a ver el significado de las propiedades con las que configuramos nuestro ComboBox:

*forceSelection*: esta opción obliga al usuario a seleccionar un valor del combo, esto es independiente del tipo de validación *allowBlank*, que discutiremos en un tema posterior.

*store*: es la fuente de datos que nuestro combo mostrará, ya hemos hablado sobre este componente.

*emptyText*: es el texto que se muestra cuando en nuestro combo no se ha seleccionado nada.

*triggerAction*: esta opción le indica al combo que siempre muestre todos los datos de su store, cuando utilizamos un store remoto aquí le pondríamos el query que se mandaría a nuestra fuente de datos remota.

*editable*: el combo no puede ser editado, es decir no se le puede escribir algún valor.

*hideTrigger*: poniendo esta propiedad en *true* hacemos que el combo aparezca sin el iconito de la flecha hacia abajo (“disparador”).

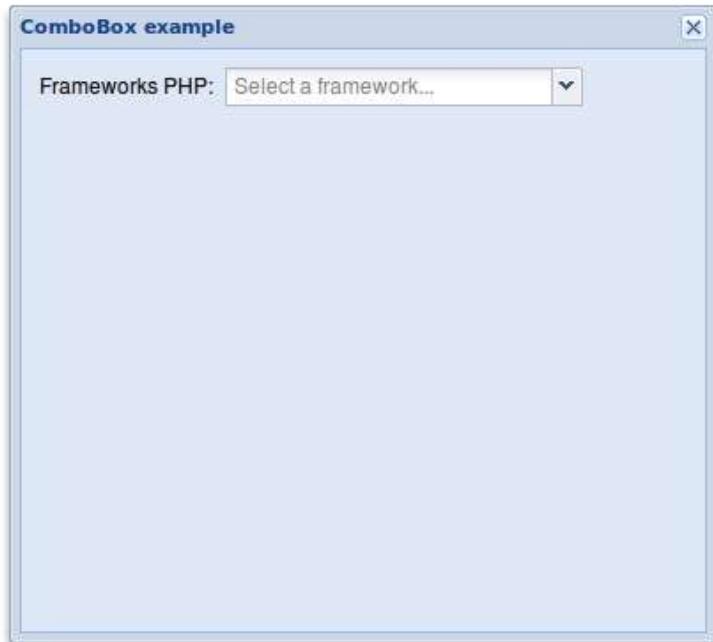
*minChars*: nos indica cuantos caracteres debemos de escribir antes de que el combo empiece a mostrar información, en nuestro caso debemos comentar la propiedad *editable* y no comentar la propiedad *minChars* para ver su funcionalidad.

Ahora lo agregamos a nuestra ventana:

```

1. var win=new Ext.Window({
2. bodyStyle:'padding: 10px',//alejamos los componentes de los bordes
3. width:400,
4. height:360,
5. items: comboLocal, //agregamos el combo a la ventana
6. layout:'form' //tipo de organización de los componentes
7. });
8. win.show();

```



Un combo cargado con información local

### ComboBox Remoto

Ahora vamos a crear un combo que cargaremos con datos de manera remota utilizando Ajax. En esta ocasión usaremos PHP, pero como ya sabemos podemos utilizar cualquier otro lenguaje servidor. En el ComboBox remoto utilizaremos para nuestra fuente de datos un store de tipo JSON, este componente no se va a detallar debido a que en un tutorial anterior se habló sobre dicho componente.

El código JS queda de la siguiente manera:

```
1. //se crea el store
2. var store= new Ext.data.JsonStore({
3. url:'combo.php',
4. root: 'data',
5. totalProperty: 'num',
6. fields: [
7. {name:'name', type: 'string'},
8. {name:'desc', type: 'string'},
9. {name:'logo', type: 'string'},
10.]
11. });
12.
13. //se crea el combo asignandole el store
14. var comboRemote=new Ext.form.ComboBox({
15. fieldLabel:'Data Base',
16. name:'cmb-DBs',
17. forceSelection: true,
18. store: store, //asignandole el store
19. emptyText:'pick one DB...',
20. triggerAction: 'all',
21. editable:false,
22. displayField:'name',
23. valueField: 'name'
24. });
```

Como podemos observar en las propiedades de nuestro combo le agregamos la propiedad *displayField*, con esta propiedad le decimos al ComboBox que información mostrar, en este caso nos va a mostrar solo

los datos 'name' y utilizando la propiedad "valueField" le especificamos cual campo del store utilizar como "value", puede ser un ID numérico pero en este caso utilizamos el nombre. Cuando usamos un store remoto podemos utilizar paginación con la propiedad 'pageSize', esta propiedad nos permite configurar el número de datos que queremos mostrar en nuestro combo. Por último es necesario agregar el combo que acabamos de crear a la ventana, de la siguiente manera:

```

1. var win=new Ext.Window({
2. title: 'ComboBox example',
3. bodyStyle:'padding: 10px', //alejamos los componentes de los bordes
4. width:400,
5. height:360,
6. items: [comboLocal,comboRemote],//se agrega el combo remoto
7. layout:'form' //tipo de organización de los componentes
8. });
9. win.show();

```



Un ComboBox cargado remotamente utilizando Ajax

Mediante código PHP vamos a exponer la información que mostrará el combo, esta información puede salir de una base de datos, o un servicio web, para poner las cosas simples y entendibles la información esta "hardcode" en arreglos.

```

1. <?php
2. $dataDB = array(
3. array(
4. "name"=>"MySQL",
5. "desc"=>"The world's most popular open source database",
6. "logo"=>"mysql.png"
7.),
8. array(
9. "name"=>"PostgreSQL",
10. "desc"=>"The world's advanced open source database",
11. "logo"=>"postgresql.png"
12.),
13. array(
14. "name"=>"Oracle",
15. "desc"=>"The world's largest enterprise software company",
16. "logo"=>"oracle.png"
17.),
18.);
19.
20. $o = array(
21. "num"=>count($dataDB),
22. "data"=>$dataDB
23.);
24. echo json_encode($o);
25. ?>

```

Nótese que se han definidos los campos que el JsonStore acepta, además mediante la función `json_decode` se está generando el JSON a partir de un arreglo de PHP.

## Datos con formato

Muy bien ahora vamos a darle formato a nuestra información, ocuparemos la información de nuestro combo remoto, así que empecemos. Vamos a crear un nuevo ComboBox que se cargará con los mismos datos que nos proporciona el PHP, ocupando el mismo store del combo anterior. Para darle formato a nuestro ComboBox ocupamos una plantilla, así que se la vamos a asignar a nuestro nuevo combo mediante una de sus propiedades ("tpl" de "template"). La plantilla está escrita en HTML con un poco de CSS para que la información se presente de una manera más detallada.

```
1. var comboRemoteTpl = new Ext.form.ComboBox({
2. fieldLabel:'Data Base',
3. name:'cmb-Tpl',
4. forceSelection:true,
5. store:store,
6. emptyText:'pick one DB...',
7. triggerAction: 'all',
8. mode:'remote',
9. itemSelector: 'div.search-item',
10. tpl: new Ext.XTemplate('<tpl for=".">><div class="search-item" style="background-
image:url({logo})"><div class="name">{name}</div><div class="desc">{desc}</div></div></tpl>'),
11. displayField:'name'
12.});
```

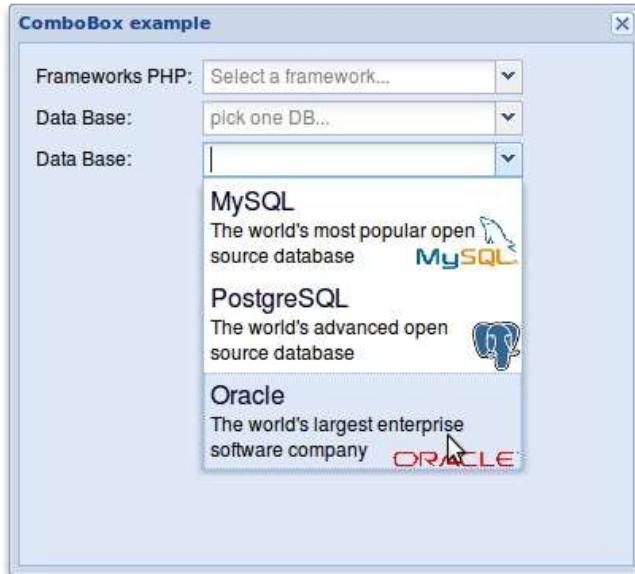
Las propiedades que aparecen ahora son:

*tpl*: le asigna un plantilla a mostrar para cada dato del combo, la plantilla que se crea es una cadena de HTML y con algunos estilos y clases de CSS, en próximos temas se dedicará un tutorial completo para analizar el funcionamiento del objeto "*Ext.XTemplate*", por ahora solo es importante mencionar que mediante un ciclo se recorre todos los registros del store y va generando la lista de opciones a presentar al usuario.

*itemSelector*: esta propiedad nos indica cual va a ser la propiedad del DOM que dispara al evento select de nuestro combo.

Hasta ahora hemos modificado el markup o HTML de la lista desplegable del combo, lo que falta es darle los estilos necesarios de la siguiente manera:

```
1. .search-item{
2. border:1px solid #fff;
3. padding:3px;
4. background-position:right right bottom bottom;
5. background-repeat:no-repeat;
6. }
7. .desc{
8. padding-right:10px;
9. }
10. .name{
11. font-size:16px !important;
12. color:#000022;
13. }
```



ComboBox personalizado con un template

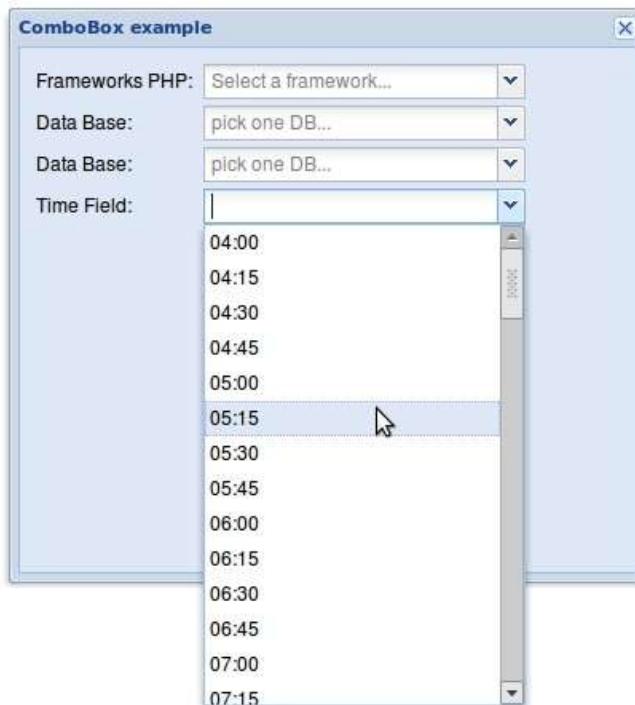
### Una variación del ComboBox

Existe un componente en ExtJs que es una variación del ComboBox este componente se llama TimeField y sirve para mostrar valores respectivos al tiempo. La mayoría de sus propiedades son las mismas que el ComboBox.

```

1. var timeField=new Ext.form.TimeField({
2. fieldLabel: 'Time Field',
3. minValue: '4:00',
4. maxValue: '23:59',
5. increment: 15,
6. format:'H:i',
7. name:'cb-time'
8. });

```



Un combo con el tiempo

Como podemos ver sin mayor complicación hemos creado un nuevo ComboBox que nos muestra datos en formatos de tiempo.

Expliquemos algunas de sus propiedades.

*minValue*: es el valor mínimo que va a mostrar nuestro campo.

*maxValue*: el valor máximo que mostrará.

*increment*: el tamaño del incremento de los valores de nuestro campo.

*format*: el formato en el que se mostrarán nuestros valores de tiempo.

### Conclusiones

El día de hoy hemos visto un componente muy completo llamado ComboBox, este componente tiene muchos diferentes tipos de configuraciones que lo hacen un componente muy recurrido en todo tipo de aplicaciones con Ext JS.

## Combos dependientes en ExtJS

En ocasiones la información que manejamos en nuestras aplicaciones está agrupada en categorías, en estas situaciones unos combos dependientes nos ayudarán a capturar estas relaciones de una manera sencilla.

En el tutorial de hoy haremos dos combos que tienen relación entre sí, usaremos el típico ejemplo de mostrar los países con sus estados, la información del combo de estados será cargada con Ajax dependiendo del país seleccionado en el primer combo, te recomiendo probar la demostración para que puedas observar el resultado final.



Ejemplo del tutorial

### Material de apoyo

Para continuar es necesario descargar el material de apoyo y copiarlo dentro de un servidor Web ya que estaremos usando Ajax para cargar la información.

### La fuente de información

Vamos a escribir el código necesario para generar la información que desplegarán los combos, para este ejemplo la información estará en arreglos de PHP, pero recuerda que puede salir de una base de datos, un servicio Web (Web Service), un archivo de texto o de cualquier otro lugar.

1. \$countries = array('Argentina','España','México','Perú','United States'); //step 1
- 2.
3. \$states = array( //step 2
4. array('Buenos Aires','Córdoba','La Pampa','Mendoza','Santa Fe'),
5. array('Asturias','Valencia','Barcelona','Toledo'),
6. array('Distrito Federal','Nuevo León','Jalisco'),
7. array('Arequipa','Puno','Tacna','Lima'),
8. array('Texas','California','New york','Virginia')
9. );

En el paso uno se han definido los países que usaremos en el primer combo, estos son cinco países, pero pudiese ser cualquier número. En el paso dos he definido los estados de cada país, es importante mencionar que la relación se está haciendo por la posición en los arreglos, debemos tener esto en mente para escribir lo siguiente:

```
1. $id = isset($_POST['id'])?$_POST['id']:-1; //step 1
2.
3. if($id > -1){ //step 2
4. //show states
5. echo toJSON($states[$id]); //step 3
6. }else{
7. //show countries
8. echo toJSON($countries); //step 4
9. }
```

En el paso uno se recoge el parámetro “id”, primero verificamos que exista, si no ha sido enviado le asignamos a la variable id un “-1”. En el paso dos verificamos que el “id” sea mayor que “-1”, esto nos dice que nuestra aplicación está solicitando los estados de un país, la variable “id” debe tener la posición del país solicitado por lo tanto aquí decidimos lo que mostraremos, países o estados.

El paso tres es ejecutado si la variable “id” contenía el índice del país seleccionado, por lo tanto desplegamos los estados del índice solicitado. El paso cuatro solamente se ejecuta si “id” es menor o igual a -1, esto nos dice que la aplicación ha solicitado los países.

Puedes ver que se está invocando a la función “toJSON” la cual definiremos a continuación:

```
1. function toJSON($array){
2. $data=array(); $i=0; //step 1
3. $total = count($array);
4. foreach($array as $key=>$value){ //step 2
5. array_push($data,array(
6. 'value'=>$i++, //step 3
7. 'label'=>$value
8.));
9. }
10.
11. return json_encode(array(//step 4
12. 'total'=>$total,
13. 'data'=>$data
14.));
15. }
```

El propósito de esta función es crear el JSON que enviaremos a los combos.

En el paso uno creamos dos variables, “data” e “i”, en la variable “data” iremos almacenando cada país o estado que desplegaremos en los combos, la variable “i” es muy importante ya que la usaremos para hacer la relación entre los combos, en el paso tres se explicará esto. En el paso dos creamos un ciclo que recorra todo el arreglo que recibimos como parámetro, en este arreglo se encuentra la información a desplegar en los combos. En el paso tres estamos almacenando en la variable “data” cada elemento del arreglo, puedes ver que se está creando un arreglo con dos propiedades “value” y “label”, estas propiedades serán usadas por el combo para su buen funcionamiento, a la propiedad “value” se le asigna el valor de “i”, este valor es el que enviaremos como parámetro para solicitar los estados del país seleccionado, la propiedad “label” simplemente es el texto que desplegará cada opción del combo.

En el paso cuatro regresamos en formato JSON la información recolectada anteriormente.

### Creación de los JsonStore

Una vez definida la información a desplegar podemos realizar los combos que usaremos, ya sabemos que los combos utilizan un Store para manipular la información que muestran.

En el material de apoyo que descargaste al inicio del tutorial viene un archivo JS (linked-cmb.js) vamos a editarla y dentro de la función “getStore” escribiremos lo siguiente:

```
1. getStore: function(){
2. var store = new Ext.data.JsonStore({
3. url:'linked-cmb.php',
4. root:'data',
5. fields: ['value','label']
6. });
7. return store;
8. }
```

Aquí solamente creamos un JsonStore, con los campos que definimos en el servidor, el código anterior deberíamos poder entenderlo pues lo hemos usado en varias ocasiones, lo que sí debo mencionar es que he decidido crear una función que me genere un Store porque necesito dos iguales, por lo tanto para no reescribir dos veces el mismo código, simplemente creé una función que me los genere las veces que sean necesarios.

### Creación de los ComboBox dependientes

En el paso anterior creamos una función que genera un JsonStore, ahora vamos a crear dos de estos para usarlos con cada uno de los combos:

```
1. this.countryStore = this.getStore();
2. this.stateStore = this.getStore();
```

El combo que desplegará los países será de la siguiente manera:

```
1. this.countryCmb = new Ext.form.ComboBox({
2. store: this.countryStore,
3. id: 'country',
4. valueField: 'value',
5. displayField: 'label',
6. triggerAction: 'all',
7. emptyText: 'Select a Country',
8. fieldLabel: 'Country'
9. });
```

Aquí no hay nada complicado, solamente una típica configuración de un ComboBox, lo siguiente que debemos hacer es crear el combo que va a desplegar los estados del país seleccionado.

```
1. this.stateCmb = new Ext.form.ComboBox({
2. store: this.stateStore,
3. disabled: true, //Step 1
4. id: 'state',
5. valueField: 'value',
6. displayField: 'label',
7. triggerAction: 'all',
8. mode: 'local', //Step 2
9. emptyText: 'Select a Country first',
10. fieldLabel: 'State'
11.});
```

Esta configuración tiene algunas diferencias con respecto a la anterior, estas diferencias son muy importantes y a continuación explico la razón:

En el paso uno estamos deshabilitando el combo, de esta manera forzamos a que el usuario seleccione primero el país, este paso es opcional pero mejora la usabilidad asegurándonos que el usuario no cometa errores en la captura. El paso dos es muy importante, si no asignamos esta configuración tendremos un

comportamiento extraño, ya que al asignarle “mode: ‘local’” nos aseguramos que no haya peticiones Ajax al servidor cuando se expanda al desplegar sus opciones, sino que las despliegue de manera “local”.

### Desplegarlos en una ventana

Hasta ahora hemos escrito código pero todavía no se ve nada en la pantalla, es hora de renderizar los componentes, para este ejemplo utilizaré una ventana de la siguiente manera:

```
1. this.window = new Ext.Window({
2. title: 'Linked ComboBox',
3. layout:'form',
4. width:300,
5. height:200,
6. bodyStyle: 'padding:5px;background-color:#fff',
7. items: [this.countryCmb,this.stateCmb]
8. });
9. this.window.show();
```



Creación de los ComboBox a utilizar

### Agregando el “listener” adecuado

Hasta este punto podemos ver los combos, el combo de países despliega su información correctamente pero el combo de estados esta deshabilitado y no podemos hacer nada con él.

Para poder interactuar con los componentes, necesitamos agregar un “listener” al primer combo, para que cuando éste se seleccione, habilite al combo de estados y cargue la información adecuada.

```
1. this.countryCmb.on('select',function(cmb,record,index){ //step 1
2. this.stateCmb.enable(); //step 2
3. this.stateCmb.clearValue(); //step 3
4. this.stateStore.load({ //step 4
5. params:{
6. id:record.get('value') //step 5
7. }
8. });
9. },this);
```

En el paso uno le agregamos un “listener” al evento “select”, este evento se dispara cuando el usuario selecciona una opción del combo, la función recibe tres parámetros: el combo, el record y el índice que ha sido seleccionado.

En el paso dos solamente habilitamos el combo de estados.

En el paso tres limpiamos el valor del combo de estados, esto permitirá que si el usuario ha seleccionado anteriormente un estado, cuando se carguen nuevos estados se limpiará el valor anterior.

En el paso cuatro estamos recargando la información del Store de estados con Ajax.

En el paso cinco estamos enviando el parámetro “id” con el cual el servidor decidirá cuales estados regresar.



Combos dependientes

### Conclusión

Siguiendo los pasos de este tutorial podemos crear combos dependientes de los niveles que necesitemos, solamente necesitamos asignar un “listener” al evento “select” del combo necesario y dentro de este “listener” recargar el store del combo relacionado.

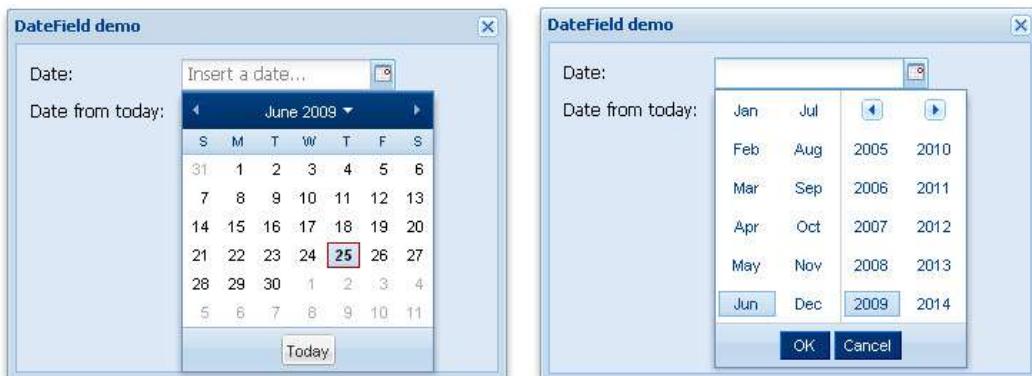
## Un calendario para capturar fechas

El día de hoy vamos a hablar del componente DateField. Este componente provee un “input” de tipo fecha con formato de un calendario muy bien estilizado. DateField en Ext JS es un componente muy completo ya que brinda muchas opciones de configuración. En este tutorial discutiremos las configuraciones más importantes.

### Material de apoyo

Para poder continuar necesitamos descargar el material de apoyo. Recuerde que este tutorial pertenece al capítulo de formularios, y al final del capítulo agregaremos nuestro DateField al formulario final, pero para evitar algunas confusiones haremos este tutorial en un archivo diferente.

Puedes ver el demo de lo que haremos en este tutorial, a continuación se muestran imágenes del componente que veremos el día de hoy.



El componente DateField

### Empaquetando el tutorial.

Procedemos a empaquetar nuestro código.

1. Ext.ns('com.quizzpot.tutorial');
2. Ext.BLANK\_IMAGE\_URL = './ext-3.0/resources/images/default/s.gif';
3. com.quizzpot.tutorial.DateFieldTutorial = {
4. init: function(){
5. //aqui va el codigo del tutorial
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorial.DateFieldTutorial.init, com.quizzpot.tutorial.DateFieldTutorial);

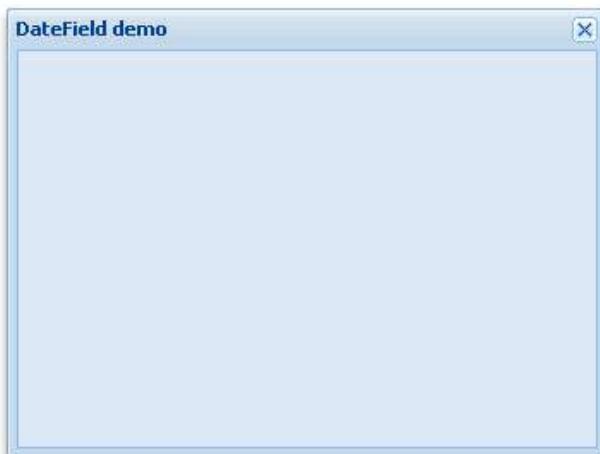
A estas alturas ya todos sabes lo que el código anterior está haciendo, de no ser así te recomiendo leer el tutorial donde se explica más detalladamente.

### Ventana

A continuación vamos a crear una ventana que alojará los diferentes tipos de configuraciones para el componente DateField que haremos, además nos servirá para visualizar los DateField que crearemos en este tutorial.

```
1. var win=new Ext.Window({
2. title: 'DateField demo',
3. bodyStyle:'padding: 10px',//alejamos los componentes de los bordes
4. width:400,
5. height:360,
6. layout:'form' //tipo de organizacion de los componentes
7. });
8. win.show();
```

Ya hemos explicado las propiedades de configuración en el tema de ventanas y paneles, si tienes dudas con el código anterior puedes repasar el tema, con respecto a la configuración “layout” hablaremos más adelante en un tema especial, por lo pronto solo ten en mente que asignándole “form” los campos se acomodarán en la ventana como en un formulario.



La ventana que contendrá los campos de fechas

### DateField Simple

Una vez creada nuestra ventana, vamos a crear nuestro primer componente tipo DateField.

```
1. var dateField = new Ext.form.DateField({
2. fieldLabel: 'Date',
3. emptyText:'Insert a date...',
4. format:'Y-m-d',
5. width: 150
6. });
```

Ahora voy a dar una descripción de las propiedades utilizadas en el código anterior:

*format*: con esta propiedad definimos el formato de la fecha que se mostrará en nuestro campo. El formato ‘Y-m-d’, define un formato del tipo “año-mes-día”. Para ver más formatos debemos consultar el API del objeto Date en Javascript.

*fieldLabel*: es la propiedad común en todos los componentes, esta propiedad define el texto que acompañará al campo y mediante esta propiedad le mostramos al usuario el tipo de información que necesita introducir en el campo, por lo tanto procura poner algún texto descriptivo.

*emptyText*: esta propiedad la hemos visto anteriormente, al usarla hacemos que el campo despliegue un texto cuando está vacío, es conveniente utilizarla para desplegar ayuda extra al usuario.

```
1. var win=new Ext.Window({
2. bodyStyle:'padding: 10px',
3. width:400,
4. height:360,
5. items: dateField, //<-- le asignamos el datefield
6. layout:'form'
7. });
8. win.show();
```



## Un DateField básico

# Un DateField Restringido

¿Qué pasa si queremos que nuestras fechas se muestren restringidas? Por ejemplo si queremos que sólo se puedan seleccionar días a partir de hoy. Para este tipo de configuración existen algunas propiedades que veremos en nuestro siguiente `DateField`.

```
1. var dateFieldR=new Ext.form.DateField({
2. fieldLabel: 'Date from today',
3. emptyText:'Insert a date...',
4. format:'Y-m-d',
5. minValue: new Date(), //<-- min date is today
6. maxValue:'2009-08-28', // <-- max date
7. value:new Date() // <-- a default value
8. });
```

## Descripción de propiedades:

*minValue*: es el valor mínimo elegible en nuestro componente DateField.

*maxValue*: es el valor máximo elegible en nuestro componente DateField.

**value:** el valor con que se inicializa el campo, por defecto es “undefined”.

Puedes ver que en este ejemplo, en la propiedad “maxValue” le he puesto directamente una fecha, esto es únicamente para demostrar que también podemos asignarle una fecha en String y el componente se encargará de hacer las debidas conversiones de datos.



Un DateField restringido

### Conclusiones

El componente DateField es un componente bastante completo y nos brinda opciones de configuración en su mayoría iguales a las del componente TextField con algunas variaciones propias del componente, esto es porque DateField hereda de TextField. Este fue tutorial corto pero al ver las diferentes configuraciones podemos comprender mejor el funcionamiento del componente DateField. En el siguiente tutorial discutiremos dos componentes muy bondadosos que son las áreas de texto y los editores HTML.

## Llenar formularios con información del servidor

En el tema de hoy veremos como llenar un formulario utilizando Ajax para realizar la petición al servidor solicitando la información mediante un identificador que mandaremos como parámetro.

Ya vimos como llenar un formulario de Ext JS a través de un record, el día de hoy lo llenaremos sacando la información a través de una petición Ajax al servidor.

### Material de Apoyo

Antes de continuar vamos a descargar el material de apoyo, el cual consta de tres archivos y una carpeta que contiene imágenes, debes copiar estos archivos al servidor Web en el que hemos estado trabajando, recuerda que esto es muy importante para que podamos utilizar Ajax.

### Demostración

En el ejercicio de hoy vamos a generar dinámicamente el “Top Ten” de las películas mas vistas en lo que va del 2009, luego al dar clic sobre alguna de estas aparecerá un formulario el cual se cargará haciendo una petición al servidor a través de Ajax, puedes ver la demostración de lo que tendremos cuando terminemos este tutorial.

|                                                                           |                              |
|---------------------------------------------------------------------------|------------------------------|
| Title:                                                                    | The Ugly Truth               |
| Year:                                                                     | 2009                         |
| Revenues:                                                                 | 27                           |
| Comment:                                                                  | Another comment to the movie |
| <input checked="" type="checkbox"/> Available                             |                              |
| <input type="button" value="Save"/> <input type="button" value="Cancel"/> |                              |

Ejemplo final

## Información

Lo más importante en una aplicación es la información, en esta ocasión (y como siempre lo hemos hecho) la información está contenida en un arreglo en el servidor, pero recuerda que puede provenir de una base de datos como MySql, PostgreSQL, Oracle, de algún servicio Web (Web Service) o de cualquier otro lugar.

```
1. <?php
2. header("Content-Type: text/plain");
3.
4. $all = isset($_GET['all']);
5. $movie = isset($_POST['id'])?$_POST['id']:rand(0,9);
6.
7. $data = array(
8. array('title'=>'G-
Force','year'=>2009,'revenues'=>32.2,'comment'=>'Very good movie, it is an awesome movie','av
ailable'=>true,'img'=>'images/gforce.jpg'),
9. array('title'=>'Harry Potter and the Half-
Blood Prince','year'=>2009,'revenues'=>30,'comment'=>'Not to good, but it is ok','available'=>true,
'img'=>'images/hpotter.jpg'),
10. array('title'=>'The Ugly Truth','year'=>2009,'revenues'=>27,'comment'=>'Another comment
to the movie','available'=>false,'img'=>'images/ugly.jpg'),
11. array('title'=>'Orphan','year'=>2009,'revenues'=>12.8,'comment'=>'Testing the comment','a
vailable'=>'images/orphan.jpg','img'=>'images/orphan.jpg'),
12. array('title'=>'Ice Age: Dawn of the Dinosaurs ','year'=>2009,'revenues'=>8.2,'comment'=>
'Awesome movie, really good','available'=>true,'img'=>'images/ice.jpg'),
13. array('title'=>'Transformers: Revenge of the Fallen','year'=>2009,'revenues'=>8,'comment'
=>'Another test','available'=>false,'img'=>'images/transformers.jpg'),
14. array('title'=>'The Hangover','year'=>2009,'revenues'=>6.46,'comment'=>'Testing','availabl
e'=>true,'img'=>'images/hangover.jpg'),
15. array('title'=>'The Proposal','year'=>2009,'revenues'=>6.42,'comment'=>'Comment','availa
ble'=>true,'img'=>'images/proposal.jpg'),
16. array('title'=>'Public Enemies','year'=>2009,'revenues'=>4.17,'comment'=>'One more com
ment','available'=>false,'img'=>'images/public.jpg'),
17. array('title'=>'Brüno','year'=>2009,'revenues'=>2.72,'comment'=>'nothing to say','available'
=>false,'img'=>'images/bruno.jpg')
18.
19.);
20.
21. if($all){
22. $info = $data;
23. }else{
24. $info = array(
25. 'success'=>true,
26. 'data'=> $data[$movie]
27.);
28. }
29.
30. echo json_encode($info);
31. ?>
```

Lo más importante a mencionar en el código anterior es lo que se imprime dependiendo del parámetro “all”, ya que si es enviado en la petición imprimirá todo el arreglo de objetos en formato “json”, si no está presente se asume que se está solicitando específicamente un elemento.

## Solicitar la información al servidor

Si en estos momentos ejecutas el HTML en tu explorador verás que solamente aparece el título, lo que tenemos que hacer es solicitar la información que vamos a desplegar utilizando Ajax, dentro del método “init” del objeto principal vamos a realizar esta petición de la siguiente manera:

```

1. Ext.Ajax.request({
2. url: 'loadform.php',
3. params: {all:true}, //solicitamos todos los registros
4. method: 'GET', //utilizando el método GET
5. scope: this,
6. success: this.createTopTen //e indicamos la función que procesará la respuesta
7. });

```

Como hemos solicitado todos los registros obtenemos el siguiente resultado:

```

1. [{"title":"G-
Force","year":2009,"revenues":32.2,"comment":"Very good movie, it is an awesome movie","available":true,"img":"images\gforce.jpg"}, {"title":"Harry Potter and the Half-
Blood Prince","year":2009,"revenues":30,"comment":"Not to good, but it is ok","available":true,"img":"images\hpotter.jpg"}, {"title":"The Ugly Truth","year":2009,"revenues":27,"comment":"Another c
omment to the movie","available":false,"img":"images\ugly.jpg"}, {"title":"Orphan","year":2009,"reve
nues":12.8,"comment":"Testing the comment","available":"images\orphan.jpg","img":"images\orp
han.jpg"}, {"title":"Ice Age: Dawn of the Dinosaurs ","year":2009,"revenues":8.2,"comment":"Aweso
me movie, really good","available":true,"img":"images\ice.jpg"}, {"title":"Transformers: Revenge of t
he Fallen","year":2009,"revenues":8,"comment":"Another test","available":false,"img":"images\tran
sformers.jpg"}, {"title":"The Hangover","year":2009,"revenues":6.46,"comment":"Testing","available
":true,"img":"images\hangover.jpg"}, {"title":"The Proposal","year":2009,"revenues":6.42,"comment
":"Comment","available":true,"img":"images\proposal.jpg"}, {"title":"Public Enemies","year":2009,"r
evenues":4.17,"comment":"One more comment","available":false,"img":"images\public.jpg"}, {"title
":"Br\u00f3nco","year":2009,"revenues":2.72,"comment":"nothing to say","available":false,"img":"ima
ges\brounco.jpg"}]

```

### **Crear la lista del “Top Ten”**

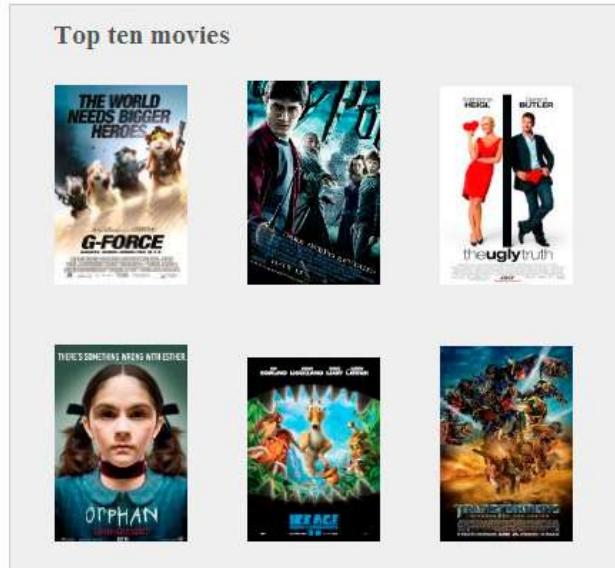
Con la información que recibimos podemos crear la lista del “Top Ten”, vamos a implementar la función “createTopTen” donde primeramente decodificamos el JSON para poder utilizarlo correctamente y luego mediante la clase “DomHelper” vamos a generar e insertar los nodos al DOM.

```

1. var info = Ext.decode(response.responseText); //decodificamos el texto que recibimos
2. Ext.each(info,function(movie){ //iteramos la información recibida
3. Ext.DomHelper.append('content',{ //y creamos una imagen para cada elemento
4. tag:'img',
5. src:movie.img,
6. alt:movie.title,
7. title:movie.title,
8. cls: 'movie'
9. });
10. },this);

```

Hemos estudiado previamente la clase “Ext.DomHelper”, por lo tanto debemos estar familiarizados con su uso, de no ser así te recomiendo repasar el tema donde estudiamos esta clase.



Creación de imágenes del Top Ten

### Agregar “listeners” a las imágenes

Lo siguiente que necesitamos hacer es agregar un “listener” al evento “clic” de cada imagen que creamos, para que podamos mostrar el formulario con la información adecuada. Justo después de crear las imágenes vamos a seleccionar los nodos del DOM, las iteramos y le agregamos el evento a cada una.

```

1. var items = Ext.DomQuery.select('div[id=content] > img');
2. Ext.each(items,function(el,i){
3. el = Ext.get(el);
4. el.on('click',function(){
5. this.showDetail(i); //esta función será disparada cuando se dé clic a una imagen
6. },this);
7. },this);

```

En el código anterior podemos ver que se disparará la función “showDetail(i);” cuando se de clic sobre alguna imagen, es importante mencionar que está recibiendo un parámetro “i” el cual indica el “id” de la imagen en la que se ha dado clic, esto nos servirá para solicitar la información correspondiente al servidor.

### Crear el formulario

Es momento de crear el formulario dentro de la función “showDetail” de la siguiente manera:

```

1. var form = new Ext.form.FormPanel({
2. url: 'loadform.php', //la URL de donde vamos a llenar el formulario
3. border:false,
4. labelWidth: 80,
5. defaults: {
6. xtype:'textfield',
7. width: 150
8. },
9. items:[
10. {fieldLabel:'Title',name:'title'},
11. {xtype:'combo',fieldLabel:'Year',name:'year',triggerAction:'all',store:[2009,2008,2007,2006]},
12. {xtype:'numberfield',fieldLabel:'Revenues',name:'revenues'},
13. {xtype:'textarea',fieldLabel:'Comment',name:'comment'},
14. {xtype:'checkbox',fieldLabel:"",labelSeparator:"",boxLabel:'Available',name:'available'}
15.]
16. });

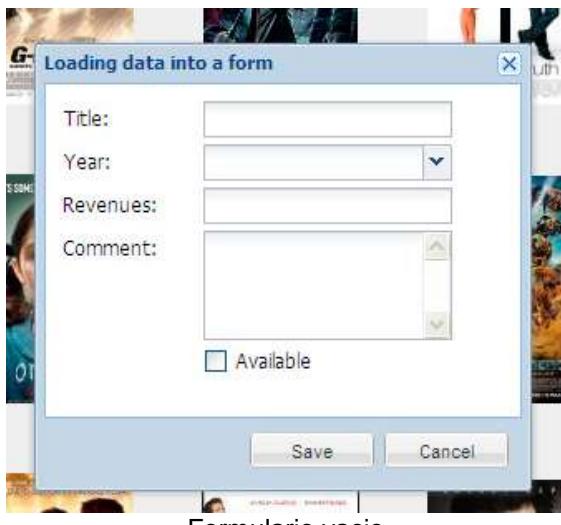
```

A estas alturas del curso debemos estar completamente familiarizados con el código anterior, lo único diferente y que es importante mencionar es la propiedad “url” en la cual se configura la “url” donde haremos la petición Ajax para llenar el formulario, además es importante notar que usamos la propiedad “name” en los campos del formulario.

Lo siguiente que debemos hacer es crear la ventana que contendrá el formulario anterior.

```
1. var win = new Ext.Window({
2. title: 'Loading data into a form',
3. bodyStyle: 'padding:10px;background-color:#fff;',
4. width:300,
5. height:270,
6. items:[form],
7. buttons: [{text:'Save'}, {text:'Cancel'}]
8. });
9.
10. win.show();
```

Si en este momento damos clic sobre cualquier imagen veremos algo semejante a la siguiente imagen.



### Solicitar la información al servidor

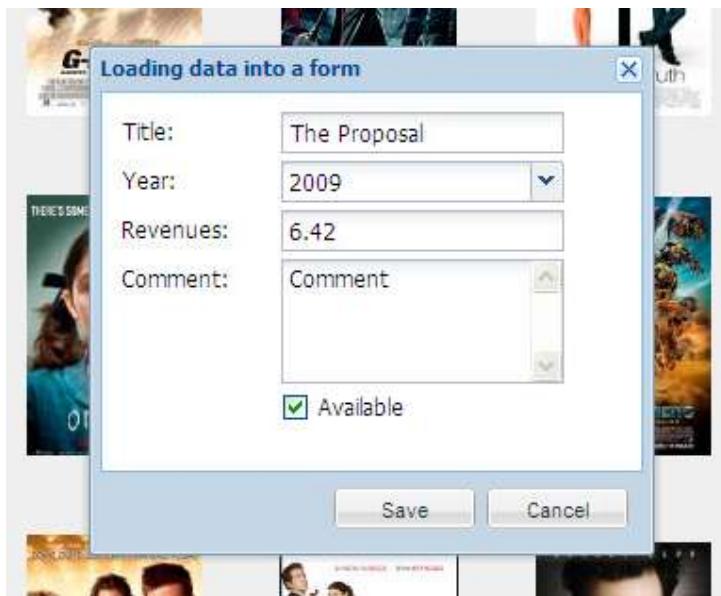
Ya creamos el formulario y aparece al momento de dar clic sobre cada imagen, lo siguiente queharemos es llenarlo con la información correspondiente a la imagen solicitada, esto es muy sencillo ya que simplemente invocamos el método “load” con los parámetros necesarios:

```
1. form.getForm().load({params:{id:id}});
```

Debemos recordar que el formato JSON que debe regresar el servidor con la información es el siguiente:

```
1. {
2. "success":true,
3. "data":{"title":"The Proposal", "year":2009, "revenues":6.42, "comment":"Comment", "available":true,
4. "img":"images\proposal.jpg"}
}
```

Es de suma importancia regresar el parámetro “success” en “true” (si no ha sucedido ningún error) para que se llene correctamente el formulario, de lo contrario el componente asume que se ocasionó algún error en el servidor y no lo llenará.



Formulario cargado mediante Ajax

### Conclusiones

Hoy aprendimos a llenar el formulario mediante una llamada Ajax al servidor, es muy importante que la respuesta contenga la propiedad "success:true" para que funcione correctamente, desde mi punto de vista (muy personal) no me agrada que el componente nos obligue a realizar esto, sería mejor utilizar algún estado http para detectar si se ha ocasionado un error, de esta manera podríamos desarrollar nuestras aplicaciones utilizando complemento "REST" (más adelante hablaré al respecto).

## Guardar información en el servidor

Hoy vamos a ver como enviar la información capturada en un formulario de Ext JS al servidor para ser guardada en una base de datos o ser procesada de la manera en que deseemos.

En los temas anteriores vimos como cargar un formulario con información en sus campos, ya sea al seleccionar una fila de un Grid o solicitando la información al servidor; en este tema mostraré como hacer un "submit" al formulario para enviar la información al servidor y de esta forma guardarla o procesarla a nuestro antojo.

### Material de apoyo

Para continuar con este tutorial es necesario descargar el material de apoyo, el cual contiene un HTML que incluye un archivo JS en el cual he creado un formulario contenido en una ventana y un archivo PHP el cual procesará la información enviada.

### Demostración

He creado una demostración del ejercicio que vamos a realizar, te recomiendo ir a probar su funcionalidad, es algo muy sencillo pero explica perfectamente el punto que estudiaremos.



Ejercicio final

## Submit tradicional

Si queremos hacer un “submit” común y corriente, es decir, sin utilizar Ajax y que toda la página se recargue, solo debemos configurar la propiedad “standardSubmit: true”, por defecto está en “false”.

Si abrimos el archivo “submitform.js”, del material de apoyo, veremos el siguiente código en el método “init”:

```
1. this.form = new Ext.form.FormPanel({
2. standardSubmit: true, // traditional submit
3. url: 'submitform.php',
4. border:false,
5. labelWidth: 80,
6. defaults: {
7. xtype:'textfield',
8. width: 150
9. },
10. items:[
11. {fieldLabel:'Title',name:'title', allowBlank:false},
12. {xtype:'combo',fieldLabel:'Year',name:'year',triggerAction:'all',store:[2009,2008,2007,2006]},
13. {xtype:'numberfield',fieldLabel:'Revenues',name:'revenues'},
14. {xtype:'textarea',fieldLabel:'Comment',name:'comment'},
15. {xtype:'checkbox',fieldLabel:"",labelSeparator:"",boxLabel:'Available',name:'available'}
16.]
17.});
18.
19. this.win = new Ext.Window({
20. id:'mywin',
21. title: 'Submit data to the Server',
22. bodyStyle: 'padding:10px;background-color:#fff;',
23. width:300,
24. height:270,
25. items:[this.form],
26. buttons: [{text:'Save'},{text:'Cancel'}]
27.});
28.
29. this.win.show();
```

Ese código lo hemos estudiado ya varias veces, por lo tanto asumo que lo conoces a la perfección; si has notado la primera configuración es “standardSubmit: true”, esta es la única diferencia con respecto a los formularios que creamos anteriormente en este curso y nos permite hacer un “submit” tradicional.

## Hacer el submit con el botón save

Hasta ahora los botones no hacen absolutamente nada, solamente están ahí de adorno, lo que tenemos que hacer es asignarle un “handler” para hacerlos funcionar y en este caso asignarle el “scope” que vamos a usar, con esto en mente cambiamos el código del botón por algo semejante a lo siguiente:

```
1. {text:'Save',handler:this.sendData,scope:this}
```

Ahora tenemos que implementar la función “sendData” para que hagamos el “submit”.

```
1. sendData: function(){
2. //submit the form
3. this.form.getForm().submit();
4. }
```

Si has notado usamos el método “getForm()” y a éste le hacemos “submit()”. El componente “FormPanel” contiene otro componente llamado “BasicForm” el cual es el responsable de manejar las funcionalidades básicas de un formulario como la información de los campos, enviar la información al servidor, resetear

los campos, validaciones, etc. A su vez el “FormPanel” es un contenedor el cual se encarga de manejar el “layout”, asignarle un título y desplegarlo en forma de un “Panel” o asignarlo a otros componentes.



Submit tradicional, no utiliza Ajax

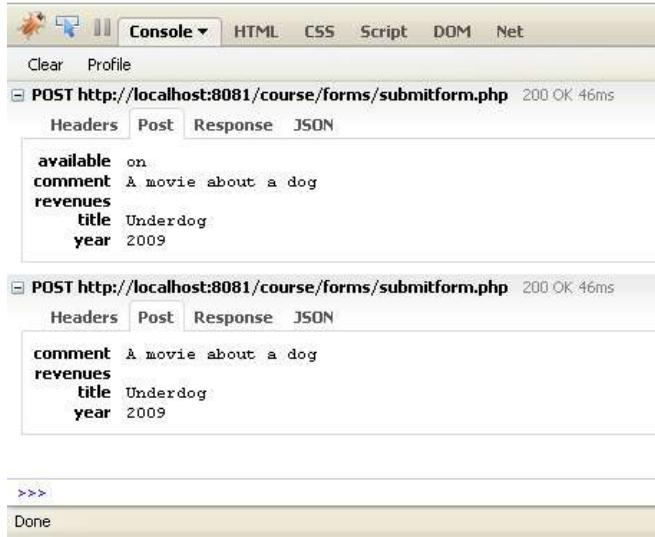
Si en este momento damos clic sobre el botón “save” veremos como se recarga completamente la página.

#### Hacer un submit utilizando Ajax

Creo que esta es la parte que a todos nos interesa, vamos a ver como podemos enviar la información utilizando Ajax. Lo primero que necesitamos hacer es quitar la configuración “standardSubmit: true” del formulario.

```
1. this.form = new Ext.form.FormPanel({
2. //standardSubmit: true, // remove this property
3. url: 'submitform.php',
4. border:false,
5. labelWidth: 80,
6. defaults: {
7. xtype:'textfield',
8. width: 150
9. },
10. items:[
11. {fieldLabel:'Title',name:'title', allowBlank:false},
12. {xtype:'combo',fieldLabel:'Year',name:'year',triggerAction:'all',store:[2009,2008,2007,2006]},
13. {xtype:'numberfield',fieldLabel:'Revenues',name:'revenues'},
14. {xtype:'textarea',fieldLabel:'Comment',name:'comment'},
15. {xtype:'checkbox',fieldLabel:"",labelSeparator:"",boxLabel:'Available',name:'available'}
16.]
17.});
```

Esto es suficiente para lograr nuestro objetivo, si probamos el ejemplo en este momento podemos ver en la consola de Firebug que se están mandando los parámetros correctamente utilizando el método “POST”.



Submit del form utilizando Ajax

### Personalizar el envío

En ocasiones es necesario cambiar el método de envío que está por defecto (POST) por algún otro, en este caso debemos especificar el que necesitemos (GET, PUT, DELETE, OPTIONS), también vamos a enviar un mensaje si se grabó satisfactoriamente la información o si se ocasionó un error.

```

1. this.form.getForm().submit({
2. method: 'put',
3. success: function(form,action){
4. Ext.Msg.alert('Success',action.result.msg);
5. },
6. failure: function(form,action){
7. switch (action.failureType) {
8. case Ext.form.Action.CLIENT_INVALID:
9. Ext.Msg.alert('Failure', 'Form fields may not be submitted with invalid values');
10. break;
11. case Ext.form.Action.CONNECT_FAILURE:
12. Ext.Msg.alert('Failure', 'Ajax communication failed');
13. break;
14. case Ext.form.Action.SERVER_INVALID:
15. Ext.Msg.alert('Failure', action.result.msg);
16. break;
17. default:
18. Ext.Msg.alert('Failure',action.result.msg);
19. }
20. }
21. });

```

En este ejemplo utilizamos el método “PUT” para crear un nuevo registro en la base de datos, ¿por qué PUT y no POST? Más adelante hablaré sobre esta técnica llamada “REST” la cual nos especifica que usemos los diferentes métodos que existen para interactuar con el servidor así como también identificar los códigos de “status” en sus respuestas. La propiedad “success” nos permite configurar una función que se ejecutará si todo salió como esperábamos, en este caso solamente mandamos un mensaje de éxito:



Grabado con éxito

Por otro lado la propiedad “failure” nos permite configurar una función a ejecutar para cuando se ocasione un error en el servidor.



Algo salió mal al grabar la información

¿Quién decide cuando se ejecuta cada método? ¡Excelente pregunta! Actualmente el componente “FormPanel” espera que la respuesta del servidor contenga la propiedad “success”, esta propiedad es la que decide cual método se ejecutará, si viene con “true” se ejecuta la función configurada en “success” y si viene en “false” se ejecuta la función configurada en la propiedad “failure”.

1. {
2. "success":true,
3. "msg":"Message example"
4. }

También se ejecuta la función “failure” cuando no se puede establecer la comunicación con el servidor o se ha ocurrido un error en la validación del formulario.

### Parámetros extra

A veces es necesario mandar parámetros extras al servidor, para esto solamente necesitamos configurar la propiedad “params” en las opciones del submit.

1. this.form.getForm().submit({
2.     method: 'put',
3.     params: {
4.         extraParam: 'Extra params!',
5.         param2: 'Param 2'
6.     },
7.     success: function(form,action){
8.         Ext.Msg.alert('Success',action.result.msg);
9.     },
10.    failure: function(form,action){
11.       switch (action.failureType) {
12.           case Ext.form.Action.CLIENT\_INVALID:
13.             Ext.Msg.alert('Failure', 'Form fields may not be submitted with invalid values');
14.             break;
15.           case Ext.form.Action.CONNECT\_FAILURE:
16.             Ext.Msg.alert('Failure', 'Ajax communication failed');

```

17. break;
18. case Ext.form.Action.SERVER_INVALID:
19. Ext.Msg.alert('Failure', action.result.msg);
20. break;
21. default:
22. Ext.Msg.alert('Failure',action.result.msg);
23. }
24. }
25. });

```

### Mensaje cargando

Para finalizar voy a mostrar como podemos enmascarar la ventana y mostrar un mensaje indicando al usuario que la información esta siendo enviada.

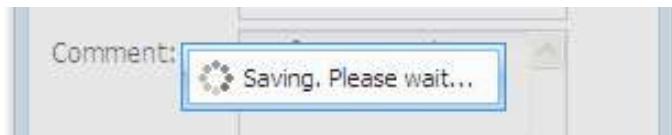
Primero necesitamos crear la mascara y mostrarla antes de realizar el submit.

```

1. var mask = new Ext.LoadMask(Ext.get('mywin'), {msg:'Saving. Please wait...'});
2. mask.show();

```

El componente “Ext.LoadMask” crea una mascara para un elemento dado, en este caso estamos asignándole la ventana que contiene al formulario, el segundo parámetro es un objeto de configuración donde le estamos asignando un mensaje a mostrar; una vez creada la “instancia” mostramos la mascara mediante el método “show”.



Máscara que muestra un mensaje al usuario

Ahora debemos ocultarla cuando el servidor nos responda, esto significa que debemos invocar el método “hide” en la función “success” y “failure”.

```

1. success: function(form,action){
2. mask.hide(); //hide the mask
3. Ext.Msg.alert('Success',action.result.msg);
4. }

```

### Conclusiones

En el tema de hoy aprendimos algunas cosas importantes, hacer un submit es algo sumamente sencillo, además de ser completamente configurable ya que se puede adaptar a nuestras necesidades.

## Validaciones simples en formularios

Las validaciones son muy importantes cuando se trata de capturar información por parte del usuario, debemos darle mucha importancia y validar toda la información que recibimos. En este tema veremos como utilizar las validaciones que vienen con Ext JS.

Es de suma importancia tener en cuenta que debemos validar la captura de información tanto del lado del cliente como en el servidor, utilizando JavaScript podemos darle una mejor experiencia al usuario para capturar información además de que ahorraremos tiempo, pero también debemos tener en cuenta que el servidor es el responsable de validar todo lo que recibe, por lo tanto también tenemos que hacer validaciones en el servidor y mostrar los errores ocasionados.

Esta es una tarea complicada, pero gracias a Ext JS podemos agilizar este proceso utilizando las validaciones que el Framework nos ofrece, en este tutorial solamente validaremos información común y sencilla, próximamente mostraré como podemos realizar validaciones más complejas o propias de nuestro negocio.

## Material de apoyo

Vamos a descargar el material de apoyo donde he creado un formulario con algunos campos comunes como son “nombre, fecha, e-mail, Website, comentario”, también incluye un archivo PHP donde aleatoriamente regresa algunos errores de validación para mostrar en los formularios.

## Demostración

Para beneficio de todos he creado un demo de lo que tendremos al final del tutorial, puedes probarlo y ver las funcionalidades que contiene.

A screenshot of a web application interface. On the left, there are three input fields with labels: "Start:" above a date input field, "Year:" above a dropdown menu, and "Email:" below them. The "Year:" field contains the value "2007". A red error message box is overlaid on the "Year:" field, containing the text "This field is required". Below the form, the text "Ejemplo del tutorial finalizado" is displayed.

## Campos requeridos

Si en este momento ejecutamos el material de apoyo vemos que aparece un formulario vacío, si damos clic sobre el botón “Save” se realiza una petición al servidor por medio de Ajax, luego nos aparece un mensaje de “éxito” o “error”, normalmente en una aplicación siempre hay campos requeridos, deberíamos validar que se envíe al servidor (por medio de Ajax) algunos campos obligatorios como el “nombre” o algunos otros.

Para definir un campo como requerido simplemente utilizamos la propiedad “allowBlank:false” en los campos que deseamos hacer obligatorios, vamos a modificar el formulario de la siguiente manera.

```
1. this.form = new Ext.form.FormPanel({
2. url: 'simplevalidations.php',
3. border:false,
4. labelWidth: 80,
5. defaults: {
6. xtype:'textfield',
7. width: 150
8. },
9. items:[
10. {fieldLabel:'Name',name:'name',allowBlank:false}, //required field
11. {xtype:'datefield',fieldLabel:'Start',name:'start',allowBlank:false}, //required field
12. {xtype:'combo',fieldLabel:'Year',name:'year',triggerAction:'all',store:[2009,2008,2007,2006],re
adOnly:true},
13. {fieldLabel:'Email',name:'email'},
14. {fieldLabel:'Web site',name:'website'},
15. {xtype:'textarea',fieldLabel:'Comment',name:'comment',allowBlank:false},//required field
16. {xtype:'checkbox',fieldLabel:"",labelSeparator:"",boxLabel:'Available',name:'available'}
17.]
18.});
```

En el código anterior solamente le he agregado la propiedad “allowBlank:false” a los campos que quiero hacer requeridos, ahora si actualizamos el explorador donde tenemos nuestro ejemplo y presionamos el botón “Save” podremos ver que nos manda un mensaje de error y nos dice pone los campos que son requeridos en color rojo:

A screenshot of a web application interface. On the left, there is an input field labeled "Comment:". A red error message box is overlaid on the input field, containing the text "This field is required". Below the form, the text "Campos requeridos en el formulario" is displayed.

Campos requeridos en el formulario

## Máximo de caracteres en un TextField

Normalmente definimos en la base de datos el numero de caracteres de algún campo, por lo tanto es recomendable asegurarnos que solamente el usuario pueda capturar hasta ese número definido, de lo contrario tendremos problemas al guardar la información, para hacer esto solamente necesitamos configurar la propiedad “maxLength” con el número máximo que necesitemos.

```
1. this.form = new Ext.form.FormPanel({
2. //... código removido para ejemplificar las líneas afectadas
3. items:[
4. {fieldLabel:'Name',name:'name',allowBlank:false,maxLength:20},
5. //... código removido para ejemplificar las líneas afectadas
6.]
7.});
```

Ahora cuando escribamos más de 20 caracteres en el campo “name” aparecerá un error que nos avisará que algo anda mal.

The screenshot shows a form panel with three fields: 'Name', 'Start', and 'Year'. The 'Name' field contains the text 'sto es algo que no debería' and has a red border. A tooltip message 'The maximum length for this field is 20' is displayed below it. The other two fields ('Start' and 'Year') are empty and have normal borders.

Restringir número de caracteres en un campo

## Validaciones comunes

Ext JS cuenta con una componente que contiene algunas validaciones comunes y que podemos agregarle las que necesitemos, en este tutorial vamos a utilizar las que vienen por defecto.

El componente responsable de almacenar estas validaciones es el “Ext.form.VTypes” y cuenta con cuatro validaciones por defecto las cuales veremos a continuación.

### Sólo caracteres alpha

Los caracteres alpha son las letras y el guión bajo, no incluye números ni cualquier otro símbolo, vamos a asignarle un “vtype” al campo “name” para que acepte solamente este tipo de caracteres.

```
1. this.form = new Ext.form.FormPanel({
2. //... código removido para ejemplificar las líneas afectadas
3. items:[
4. {fieldLabel:'Name',name:'name',allowBlank:false,maxLength:20,vtype:'alpha'},
5. //... código removido para ejemplificar las líneas afectadas
6.]
7.});
```

Mediante la propiedad “vtype” podemos configurar la validación que necesitemos, en este caso utilizamos “alpha” la cual esta definida dentro del componente “Ext.form.VTypes”, esto es importante tomarlo en cuenta.

The screenshot shows a form panel with three fields: 'Name', 'Start', and 'Year'. The 'Name' field contains the text 'Crysfel's' and has a red border. A tooltip message 'This field should only contain letters and \_' is displayed below it. The other two fields ('Start' and 'Year') are empty and have normal borders.

Sólo caracteres alpha

### Validar un correo electrónico (e-mail)

Una validación muy común es el e-mail, vamos a asignarle el “vtype” correspondiente al campo del nuestro formulario de la siguiente manera:

```

1. this.form = new Ext.form.FormPanel({
2. //... código removido para exemplificar las líneas afectadas
3. items:[
4. //... código removido para exemplificar las líneas afectadas
5.
6. {fieldLabel:'Email',name:'email',vtype:'email'}, // validar un e-mail
7.
8. //... código removido para exemplificar las líneas afectadas
9.]
10. });

```

Gracias a Ext JS no demoramos ni un minuto en agregar esa validación, ahora el formulario se verá semejante a la siguiente pantalla cuando introduzcamos un e-mail inválido.

Validación de un e-mail

### Validar un URL

Vamos a ver como validar una dirección o “URL” de un sitio Web.

```

1. this.form = new Ext.form.FormPanel({
2. //... código removido para exemplificar las líneas afectadas
3. items:[
4. //... código removido para exemplificar las líneas afectadas
5.
6. {fieldLabel:'Web site',name:'website',vtype:'url'}, //validar una URL
7.
8. //... código removido para exemplificar las líneas afectadas
9.]
10. });

```

Es importante notar que también acepta protocolos “https” y “ftp” para validar correctamente.

Validación de una URL

### Validar en el servidor

Hasta ahora hemos visto como validar en el cliente, pero también es de suma importancia validar en el servidor, vamos a ver como podemos desplegar los errores de validación que se ocasionan en el servidor.

Ext JS cuenta con una funcionalidad que nos ahorrará mucho tiempo al desplegar los errores ocasionados en el servidor, marcando el campo exacto donde se ocasionó y desplegando un mensaje del porqué del error, para poder utilizar esta funcionalidad únicamente la respuesta del servidor debe estar en el siguiente formato:

1. {

```

2. success: false,
3. msg: 'Mensaje general de error',
4. errors: {
5. campo : 'Mensaje del porqué',
6. otroCampoConError: 'porque se generó el error'
7. }
8. }

```

Teniendo en mente lo anterior vamos a editar el archivo "simplevalidations.php" y generar algunos errores de prueba.

```

1. $info = array(
2. 'success' => false,
3. 'msg' => 'There was an error saving the record',
4. 'errors' => array(
5. 'name' => '$name+' is not an employee of this corporation',
6. 'email' => 'E-mail does not exist in the database',
7. 'comment' => 'The comment is too short'
8.)
9.);

```

Estos cambios son suficientes, así que no tenemos que hacer nada más, veamos el resultado.

|        |                     |
|--------|---------------------|
| Name:  | Crysfel             |
| Start: | 08/24/2009          |
| Year:  | 2009                |
| Email: | user.name@gmail.com |

Desplegar errores de validación ocasionados en el servidor

Con esto podemos ver que aunque las validaciones del cliente sean correctas, la información puede ser incorrecta debido a nuestra lógica de negocio, en este ejemplo el nombre de la persona debería existir en nuestro catálogo de empleados (ejemplo hipotético).

### Conclusiones

Las validaciones son muy importantes, hoy vimos como utilizar las validaciones por defecto, además aprendimos como desplegar los mensajes de error que se generan en el servidor, con Ext JS todo esto es más sencillo.

## Validaciones personalizadas

Muchas veces es necesario hacer validaciones propias de nuestra aplicación y que son necesarias para el correcto funcionamiento de nuestro sistema, en este tema se explica como podemos crear "vtype's" para validar nuestros formularios de Ext JS.

En el tutorial anterior vimos como hacer validaciones comunes, el día de hoy vamos aprender como hacer validaciones personalizadas además veremos como restringir la entrada de caracteres en las cajas de texto.

Te invito a probar la demostración de lo que haremos en este tutorial.

|                |                     |
|----------------|---------------------|
| Lastname:      | Doe                 |
| Date of birth: | 08/03/1995          |
| Phone number:  | (82) 36 128 362     |
| Credit card:   | 1234 5678 0012 3456 |

Formulario con validaciones personalizadas

## Material de apoyo

Antes de seguir es necesario descargar el material de apoyo, el cual consta de dos archivos solamente, un HTML y un JS; he creado un formulario y lo he puesto dentro de una ventana, así que el ejercicio será agregar validaciones para algunos campos.

### Mostrar los mensajes de error

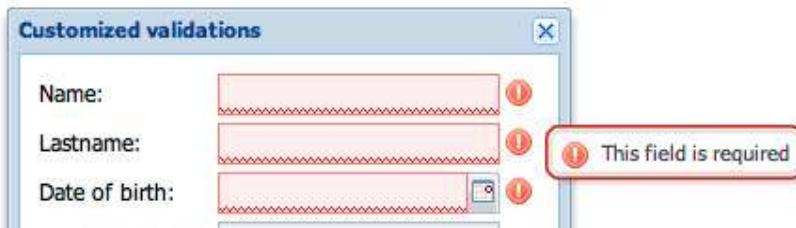
Si en este momento ejecutamos el material de apoyo veremos que tres campos son requeridos, y al presionar el botón “save” nos dice que existen errores (si no ponemos nada en las cajas de texto), pero únicamente se está poniendo en color rojo el contorno de los campos.



El usuario debe recibir suficiente información para saber que está mal y cómo solucionarlo, por lo tanto lo que necesitamos hacer es mostrar el mensaje del error ocasionado, para esto solamente necesitamos escribir las siguientes líneas al inicio del método “init”:

1. Ext.QuickTips.init();
2. Ext.form.Field.prototype.msgTarget = 'side';

La primera línea permite que aparezcan los mensajes flotantes cuando posicionamos el Mouse sobre la caja de texto, con la segunda línea de indicamos que aparezca un ícono de error en la parte lateral del campo.



Desplegamos un mensaje flotante con el error de validación

### Validar mayoría de edad

En el campo “Date of birth” vamos a validar que dada la fecha de nacimiento calculemos cuantos años hay hasta el día de hoy, luego verificamos que sea mayor de 18 para pasar la validación del campo.

En el tema anterior vimos como utilizar los “vtype’s” que vienen incluidos en el framework de Ext JS, hoy vamos agregar algunas validaciones más a la clase “Ext.form.VTypes”.

Lo primero que necesitamos hacer es extender la clase mencionada de la siguiente manera:

1. Ext.apply(Ext.form.VTypes,{
- 2.
3.     //aqui vamos a definir las validaciones necesarias
- 4.
- 5.});

Lo siguiente es definir el “vtype” de la siguiente manera:

```

1. Ext.apply(Ext.form.Validators,{
2. adult: function(val, field){
3. try{
4. var birth = field.getValue();
5. var now = new Date();
6. // The number of milliseconds in one year
7. var ONE_YEAR = 1000 * 60 * 60 * 24 * 365;
8. // Convert both dates to milliseconds
9. var date1_ms = birth.getTime()
10. var date2_ms = now.getTime()
11. // Calculate the difference in milliseconds
12. var difference_ms = Math.abs(date1_ms - date2_ms)
13. // Convert back to years
14. var years = difference_ms/ONE_YEAR;
15. return years >= 18;
16. }catch(e){
17. return false;
18. }
19. },
20. adultText: 'You are underage!', //mensaje de error
21. adultMask: // //regexp para filtrar los caracteres permitidos
22. });

```

Primero se define el nombre que le daremos al “vtype”, en este caso “adult” luego le asignamos una función que recibe dos parámetros, el contenido del campo y la instancia del campo que se está validando, dentro de esta función pondremos la lógica para validar y debe regresar “true” para cuando el valor capturado sea correcto o “false” para cuando el valor del campo sea incorrecto.

Lo siguiente que debemos hacer es definir el texto que se mostrará en caso de que la información capturada sea incorrecta, debemos recordar que existen más validaciones dentro de este objeto (Ext.form.Validators), por lo tanto para relacionar el mensaje con la función de validación simplemente le agregamos la palabra “Text” al final del nombre la función, en este caso “adultText”.

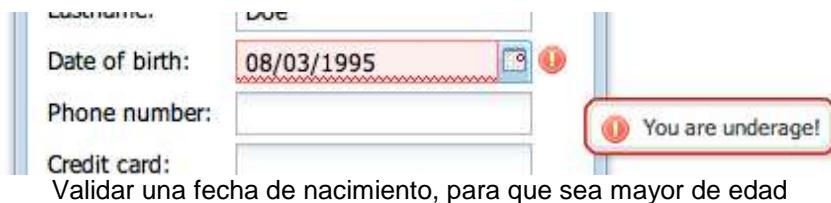
Opcionalmente podemos definir un filtro para capturar solamente los caracteres correctos, para eso necesitamos definir una máscara con la expresión regular adecuada.

- adultMask: /\d{V}/

La expresión regular anterior valida que solamente se puedan capturar dígitos y una diagonal, estos caracteres son los únicos que necesitamos para escribir una fecha en el formato que hemos definido, nótese que la propiedad se llama “adultMask”, solamente le agregamos al final la palabra “Mask” para relacionarla con la validación que creamos anteriormente.

Una vez definida el “vtype” necesitamos asignárselo al campo que necesitamos validar, en este caso al campo “birth” del formulario:

- {xtype:'datefield',fieldLabel:'Date of birth',name:'birth',allowBlank:false,format:'d/m/Y',vtype:'adult'},



### Validar un teléfono

Vamos a validar el formato de un número telefónico, primero necesitamos definir las reglas de validación.

- Tiene que ser de 10 dígitos, ejemplo: 8192847135

- Solamente puede aceptar paréntesis, guiones, espacios y números, ejemplos: (81) 92 847 135, 81 92-847-135, (81) 92-847-135, 81 92 84 71 35, 8192847135
- No debe capturar ningún otro carácter de los definidos anteriormente

Con esas reglas en mente definimos el “vtype” correspondiente:

1. phone: function(value,field){
2. return value.replace(/[\-\(\)]/g,"").length == 10;
3. },
4. phoneText: 'Wrong phone number, please make sure it contains 10 digits',
5. phoneMask: /\d\-\(\)/

La función definida en la propiedad “phone” solamente remueve los caracteres aceptados (espacios, guiones, paréntesis) dejando solamente los números, luego valida que sean diez dígitos, la máscara solo acepta los caracteres definidos en las reglas.

Por último necesitamos asignarle el “vtype” al campo del formulario que necesitamos validar.

1. {fieldLabel:'Phone number',name:'phone',vtype:'phone'}

The screenshot shows a user interface for validating a phone number. At the top, there is a status bar with the text "Phone number: 81 12 34 543". Below this, there are three input fields: "Phone number" containing "81 12 34 543", "Credit card" (empty), and "Comment" (empty). To the right of the "Phone number" field, a red-bordered box contains the error message "Wrong phone number, please make sure is 10 digits". Below the form, the text "Validar un número telefónico" is visible.

### Validar una tarjeta de crédito

En este último ejemplo vamos a validar el número de una tarjeta de crédito, primero necesitamos definir las reglas de validación para tener una idea clara de lo que haremos.

- El número debe ser de 16 dígitos, ejemplo: 1234567891234567
- Solo puede aceptar dígitos, espacios y guiones, ejemplo: 1234 5678 9123 4567, 1234-5678-9123-4567
- No debe aceptar ningún otro carácter que de los anteriores mencionados

Si te das cuenta estas reglas son muy semejantes a las del teléfono, así que podemos hacer un copy/paste y modificar el código para que se adapte a estos requerimientos.

1. creditcard: function(value,field){
2. return value.replace(/[\-\]/g,"").length == 16;
3. },
4. creditcardText: 'Wrong credit card number',
5. creditcardMask: /\d\-\/

No cambié muchas cosas en esta parte ya que las reglas son muy semejantes, ahora necesitamos agregar el “vtype” al campo “credit”.

1. {fieldLabel:'Credit card',name:'credit',vtype:'creditcard'}

The screenshot shows a user interface for validating a credit card number. At the top, there is a status bar with the text "Credit card: 1234 5678 9012 345". Below this, there are three input fields: "Credit card" containing "1234 5678 9012 345", "Comment" (empty), and another "Credit card" field (empty). To the right of the "Credit card" field, a red-bordered box contains the error message "Wrong credit card number". Below the form, the text "Validar una tarjeta de crédito" is visible.

## Conclusiones

El día de hoy aprendimos a crear nuestras propias validaciones del lado del cliente, como puedes ver fue realmente sencillo. Ext JS nos provee una manera muy fácil de lograr validar e inclusive poner una máscara a nuestros campos.

## Campos Compuestos

En el tutorial de hoy se explica cómo se puede crear más de un campo editable en una sola fila, esto es un caso muy típico cuando capturamos campos como un teléfono, una tarjeta de crédito, etc.

Por medio del componente “CompositeField” podemos separar la información de un campo en varios textfields en una misma fila, esto nos ayuda a mejorar la interfaz de nuestra aplicación.

Esta es una muestra de lo que se obtendrá al final de este tutorial. Recuerda que puedes descargar el código fuente si es necesario.

The screenshot shows a window titled "Composite Form". Inside, there are four input fields: "Name" (text), "Lastname" (text), "E-mail" (text), and "Phone" (text). The "Phone" field contains a placeholder "(   ) -   -   ". A "Save" button is located at the bottom right of the form area.

Resultado Final

## La base de datos

Para este ejemplo usaremos una base de datos solo para guardar los nuevos registros que se creen. La base de datos solo contiene una tabla llamada “friends” que es donde se guardará la información. Actualmente se encuentra vacía.

```
1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Servidor: localhost
6. -- Tiempo de generación: 20-10-2010 a las 23:14:21
7. -- Versión del servidor: 5.1.36
8. -- Versión de PHP: 5.3.0
9.
10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11.
12. --
13. -- Base de datos: `test`
14. --
15. -----
16. --
17. -- Estructura de tabla para la tabla `friends`
18. --
19. CREATE TABLE IF NOT EXISTS `friends` (
20. `id` int(11) NOT NULL AUTO_INCREMENT,
21. `name` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
22. `lastname` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
23. `email` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
24. `phoneNumber` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
25. PRIMARY KEY (`id`)
```

```

26.) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT
 =10 ;
27. --
28. -- Volcar la base de datos para la tabla `friends`
29. --

```

La base de datos se llama “test” pero puedes usar el nombre que te agrade, solo recuerda cambiar el nombre cuando se haga la conexión mediante PHP.

### **Empaquetando el tutorial**

Vamos a encapsular el código para evitar conflictos con otras variables. Esto es algo muy importante y siempre lo debemos realizar.

```

1. Ext.ns("com.quizzpot.tutorials");
2.
3. com.quizzpot.tutorials.compositeForm ={
4. init : function(){
5. //code...
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorials.compositeForm.init,com.quizzpot.tutorials.compositeForm);

```

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### **Creando el formulario**

Necesitamos un formulario por el cual se obtendrá la información de algún contacto del usuario. Lo primero que haremos es crear este formulario el cual contiene cuatro campos: nombre, apellido, correo electrónico y teléfono. Lo haremos de la siguiente manera:

```

1. this.form = new Ext.FormPanel({ step 1
2. url : 'submit.php', //step 2
3. bodyStyle : "padding : 10px",
4. border : false,
5. defaults : {anchor: "100%", allowBlank: false},
6. items : [//step 3
7. {xtype : "textfield", name : "first", fieldLabel : "Name", labelWidth : 20},
8. {xtype : "textfield", name : "last", fieldLabel : "Lastname"},
9. {xtype : "textfield", name : "email", fieldLabel : "E-mail", vtype : "email"},
10. this.compositeFields ()],
11. });

```

En el paso uno creamos el formulario, esto es algo que ya hemos visto en tutoriales anteriores.

En el paso dos se le pasa el parámetro “url” donde se realizará el submit del formulario con la información capturada.

En el paso tres creamos los “textfield” de manera “lazy” por medio de su “xtype”, estos los usaremos para obtener la información que nos dará el usuario. Los primeros tres campos los creamos como lo hemos hecho siempre, notemos que el tercer campo cuenta con un “vtype”, esto lo hacemos con la intención de verificar que el campo tenga la estructura básica de un correo electrónico.

Si notamos al final del arreglo de “items” tenemos la función “compositeFields” la cual veremos en unos momentos. Así que la podemos comentar por el momento.

Ya que tenemos nuestro formulario lo siguiente es crear la ventana donde lo mostraremos al usuario, lo haremos de la siguiente manera:

```

1. var win = new Ext.Window({
2. title : "Composite Form",

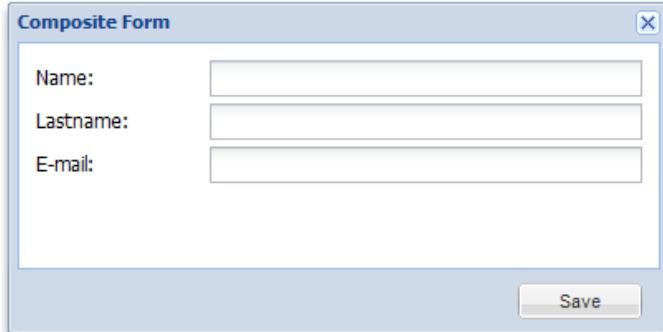
```

```

3. layout : "fit",
4. width : 400,
5. height : 200,
6. items : this.form,
7. fbar : [{text: "Save"}]
8.
9. });
10. win.show();

```

La creación de una ventana ya es algo que se ha tratado en temas anteriores, por lo cual no entraremos en detalles. Pero si haremos notar que se le está colocando el formulario en el atributo “ítems”, también estamos creando un “fbar” que contiene el botón “Save”. Con esto tendríamos algo como lo que se muestra a continuación:



Colocando el Formulario

### Función compositeFields

En esta función veremos cómo implementar los “CompositeField”, lo que esto nos permite es tener una cierta cantidad de campos de texto en una sola fila. Esto es posible a partir de la versión 3.X de Ext JS, a continuación veremos la implementación de esto.

```

1. compositeFields : function(){ //step 1
2. return{
3. xtype : "compositefield", //step 2
4. fieldLabel : "Phone",
5. defaults : {allowBlank: false},
6. border : false,
7. items : [{xtype : "displayfield", value:"("}, //step 3
8. {xtype : "textfield", name : "phoneNum1", width: 30}, //step 4
9. {xtype : "displayfield", value:") - "},
10. {xtype : "textfield", name : "phoneNum2", width: 50},
11. {xtype : "displayfield", value:" - "},
12. {xtype : "textfield", name : "phoneNum3", width: 50}
13.]
14. };
15. },

```

En el paso uno creamos la función.

En el paso dos creamos el componente “compositefield” por medio de su “xtype”, es importante mencionar que esto también se puede hacer realizando una instancia de “Ext.form.CompositeField” con el operador “new”.

En el paso tres usamos un “displayfield” para mostrar algo de texto antes del “textfield”.

Del paso cuatro en adelante se crean los “textfield” que se usarán para recibir la información del usuario. Notemos que entre cada “textfield” hay un “displayfield” esto es con el objetivo de dotar de mayor usabilidad al formulario.

Con esto tendríamos algo como lo siguiente:

The screenshot shows a window titled "Composite Form". It contains four text input fields: "Name", "Lastname", "E-mail", and "Phone". The "Phone" field is a composite field consisting of three smaller input fields for area code, exchange, and number. A "Save" button is located at the bottom right of the window.

Completoando el Formulario

Como podemos notar al usar un “CompositeField” podemos tener varios “textfields” en una sola fila, a diferencia de como siempre lo hemos hecho, colocándolos en diferentes filas.

### Enviando la información

Ya que tenemos nuestro formulario listo, lo siguiente es guardar la información en la base de datos, lo primero que faremos será agregar un handler al botón “Save”, de esta manera podemos agregarle acciones para cuando el usuario de clic sobre este.

```
1. var win = new Ext.Window({
2. title : "Composite Form",
3. layout : "fit",
4. width : 400,
5. height : 200,
6. items : this.form,
7. fbar : [{text: "Save",handler:this.onSave,scope:this}] //step 1
8.
9. });
10. win.show();
```

En el paso uno agregamos el “handler” al botón, esta función se dispara cada vez que se da clic sobre el botón. En este ejemplo el “handler” tiene definida la función “onSave” la cual se encarga de hacer la petición Ajax al servidor enviando la información introducida por el usuario.

```
1. onSave :function(){
2.
3. this.form.getForm().submit({
4. scope : this,
5. success : this.showResults,
6. failure : function(response){
7. console.debug(response);
8. }
9. });
10. },
```

Es importante mencionar que en el atributo “success” se emplea otra función que es showResults, lo que hace esta función es simplemente mandar un mensaje de alerta informándonos que todo salió bien, aquí también usamos la configuración “scope”, esto permite definir el contexto donde se ejecutará la función asignada al callback “success” y “failure”.

### Guardado la información

En los pasos anteriores realizamos el “submit” al formulario una vez que el usuario da clic sobre el botón “Save”, ahora vamos a guardar la información en la base de datos. En el archivo “submit.php” escribimos el siguiente código:

```

1. <?php //step 1
2. $connection=mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
4.
5. //step 2
6. $name = $_POST["first"];
7. $last = $_POST["last"];
8. $email = $_POST["email"];
9. $phone1 = $_POST["phoneNum1"];
10. $phone2 = $_POST["phoneNum2"];
11. $phone3 = $_POST["phoneNum3"];
12. //step 3
13. $phone = $phone1.'-'.$phone2.'-'.$phone3;
14.
15. //Se recomienda limpiar aquí las variables de entrada antes de guardarlas en a base de datos!!
16.
17. //step 4
18. $query = sprintf("INSERT INTO friends(name,lastname,email,phoneNumber) values('%s','%s',
19. '%s','%s')",
20. mysql_real_escape_string($name),
21. mysql_real_escape_string($last),
22. mysql_real_escape_string($email),
23. mysql_real_escape_string($phone));
24. $rs = mysql_query($query);
25.
26. echo json_encode(array(
27. "success" => true
28.));

```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”. En el paso dos recibimos vía POST la información introducida por el usuario, esta información fue enviada automáticamente al hacer “submit” al formulario que definimos en ExtJS. En el paso tres lo que hacemos es concatenar las tres partes del número telefónico, para que sea guardado como una sola cadena de caracteres. En el paso cuatro introducimos la información a la tabla “friends”. Es importante mencionar que se deben limpiar las variables antes de insertarlas a la base de datos, esto para evitar ataques XSS, existen librerías que te permiten realizar la limpieza de una manera sencilla.

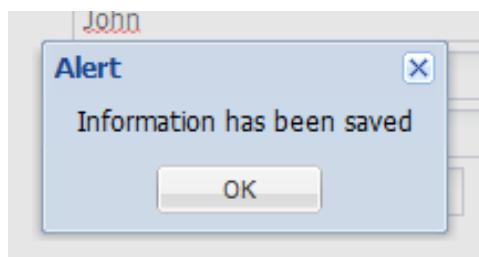
### Mensaje de éxito

Una vez que se ha guardado el nuevo contacto crearemos la función “showResult” que es el callback para cuando la comunicación ha sido satisfactoria, dentro de esta función solamente mandaremos un mensaje indicándole al usuario que su información ha sido guardada.

```

1. showResults : function(response){
2. Ext.Msg.alert("Alert"," Information has been saved!");
3. }

```



Mensaje de Éxito

## Conclusión

En este tutorial vimos un uso básico del “CompositeField”, este componente nos ayuda a tener más de un “textfield” en la misma fila, esto es conveniente cuando queremos tener una cadena de caracteres dividida en varios campos editables, como el número telefónico en este ejemplo.

## Gráficas

Uno de los nuevos componentes de ExtJS 3.0 nos permite crear gráficas de diferentes tipos, en este capítulo mostramos algunas

### Gráficas en Ext JS 3

Una de las funcionalidades nuevas que tiene la versión tres de la librería de Ext JS es la posibilidad de crear fácilmente gráficas de barras, lineares y de pastel a partir de un store.

#### Material de apoyo

Para continuar es necesario descargar el material de apoyo, el cual contiene un HTML que únicamente está incluyendo la librería de Ext JS (la versión 3 RC 1) y un JS vacío en el cual escribiremos el código necesario para generar las gráficas.

#### Empaquetar el tutorial

Antes de seguir adelante vamos a “empaquetar” el código que estaremos escribiendo, ya sabemos las ventajas de hacerlo ya que estudiamos un tema al respecto.

```
1. //the namespace for this tutorial
2. Ext.ns('com.quizzpot.tutorial');
3.
4. com.quizzpot.tutorial.Charts = {
5. init: function(){
6. //Aquí vamos a escribir el resto del código
7. }
8. }
9.
10. Ext.onReady(com.quizzpot.tutorial.Charts.init,com.quizzpot.tutorial.Charts);
```

El código anterior debe ir en el archivo “chart.js”

#### Definir la información a graficar

Ahora vamos a definir la información que necesitamos graficar, para este ejemplo voy a graficar diferentes librerías de Javascript y el número de usuarios tienen sus comunidades (la información es totalmente falsa, una suposición solamente), es importante mencionar que esta información puede venir de una base de datos o de alguna otra fuente, además puede ser interpretada en cualquier formato soportado por el store (XML, JSON, Array), para este ejemplo la información proviene de un arreglo definido directamente en el código.

```
1. //información a graficar
2. var data = [['Ext JS',115000],['jQuery',250100],['Prototype',150000],['mootools',75000],['YUI',95000]
 ,['Dojo',20000],['Sizzle',15000]];
3.
4. //se crea el Store que manipula la información
5. var store = new Ext.data.ArrayStore({
6. fields:[{name:'framework'}, {name:'users', type:'float'}]
7. });
8. store.loadData(data); // se carga la info en el store
```

El código anterior ya lo hemos discutido en temas anteriores por lo tanto debe ser familiar.

#### Las gráficas

Este componente es una adaptación de la librería YUI, la cual funciona utilizando un “swf” (una película de flash) para poder generar la imagen de la gráfica, es “personalizable” pues nos permite cambiar el

aspecto de la gráfica (colores, tipografía, estilos), los componentes se encuentran dentro del paquete "Ext.chart" así que te recomiendo mirar la documentación.

### Gráfica de barras

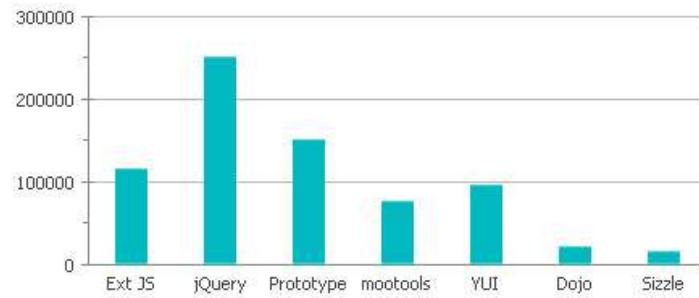
Vamos a crear una gráfica de barras con la información que tenemos en el store, esto lo hacemos de la siguiente manera:

```
1. var columnChart = new Ext.chart.ColumnChart({
2. store: store,
3. url:'..../ext-3.0-rc1/resources/charts.swf',
4. xField: 'framework',
5. yField: 'users'
6. });
```

Las propiedades básicas de configuración son esas tres solamente, adicionalmente podemos especificarle la "url" donde se encuentra el "swf" que se encarga de mostrar la gráfica, esto es importante si no queremos que lo busque directamente en la página de Yahoo y es obligatorio si no contamos con acceso a Internet.

Mediante la propiedad "xField" se le indica a la gráfica de donde tomar la información del eje "X", mientras que la propiedad "yField" indica la fuente de información para el eje "Y", estos dos campos son necesarios para generar una gráfica de barras.

Si le agreamos la propiedad "renderTo", podemos renderear en pantalla la gráfica y se verá de la siguiente manera



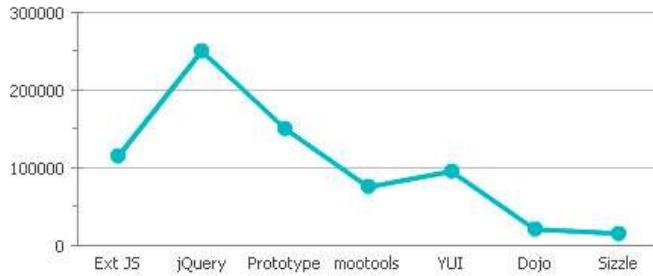
Gráfica de barras

### Gráfica lineal

Una gráfica lineal se hace exactamente igual que una gráfica de barras, únicamente utilizamos el componente "LineChart" de la siguiente manera.

```
1. var lineChart = new Ext.chart.LineChart({
2. store: store,
3. xField: 'framework',
4. yField: 'users'
5. });
```

Con eso es suficiente para generar una gráfica lineal muy básica, para renderearla en pantalla podemos usar la propiedad "renderTo".



Gráfica lineal

### Gráfica de pastel (Pie Chart)

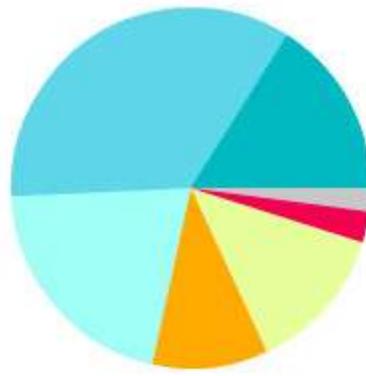
La gráfica de pastel se crea diferente a las anteriores, y esto es porque aquí no hay ejes (x | y), una gráfica de pastel se crea por porcentajes, el componente “PieChart” se encarga de todo, únicamente es necesario configurar lo siguiente:

```

1. var pieChart = new Ext.chart.PieChart({
2. store: store,
3. dataField: 'users', //la información a graficar
4. categoryField : 'framework' //las etiquetas o categorías
5. });

```

En el código anterior la propiedad “dataField” es la que contiene la información a graficar y la propiedad “categoryField” es la que contiene las categorías que se están graficando, estas dos propiedades son muy importantes para que la gráfica se pueda generar correctamente.



Gráfica de pastel

### Colocarlas en pantalla

En este paso vamos a crear los paneles necesarios, se los asignamos a uno principal y lo “renderizamos” en el div “frame” que se encuentra en el documento HTML, además haremos que los paneles que contienen la gráfica se colapsen.

```

1. var panel1 = new Ext.Panel({
2. title: 'Column chart example',
3. items:[columnChart]
4. });
5.
6. var panel2 = new Ext.Panel({
7. title: 'Line chart example',
8. items:[lineChart]
9. });
10.
11. var panel3 = new Ext.Panel({
12. title: 'Pie chart example',

```

```

13. items:[pieChart]
14. });
15.
16. var main = new Ext.Panel({
17. renderTo: 'frame',
18. width:450,
19. defaults: {
20. height:250,
21. collapsible: true,
22. border:false,
23. titleCollapse: true
24. },
25. items: [panel1,panel2,panel3]
26. });

```

El código que hemos escrito en éste tutorial genera la siguiente pantalla.



Gráficas en Ext JS 3

Hemos visto anteriormente el funcionamiento de los paneles así que debes estar familiarizado con el código anterior.

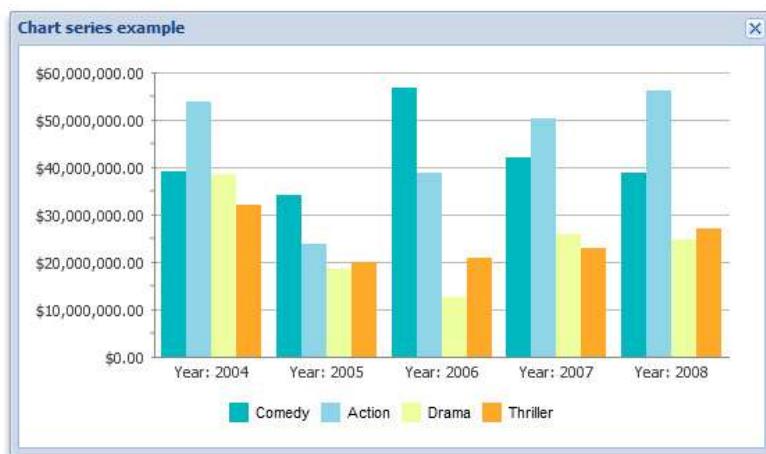
### Conclusiones

Las gráficas que nos proporciona la versión tres de Ext JS, nos permiten desplegar la información de una manera muy sencilla y agradable, además de que se integra muy fácilmente a los otros componentes (paneles, ventanas, formularios, etc.).

## Graficando series

Anteriormente hemos hablado acerca del nuevo componente “Chart”, aprendimos como crear una gráfica sencilla circular o de “pastel”, una linear y de columnas, el día de hoy veremos como crear una gráfica con una serie de datos.

En este tutorial vamos a graficar una serie que nos ayude a visualizar la diferencia de “ganancias” en un determinado año de cierto “Género” de película, esto lo graficaremos en un mismo “Plano cartesiano”, te recomiendo probar la demostración que he preparado.



Graficando series

### Material de apoyo

Antes de continuar es necesario descargar el material de apoyo donde únicamente viene un HTML y un archivo JS.

### Definir la información a graficar

Lo primero que necesitamos hacer es definir de donde tomaremos la información que vamos a graficar, para fines prácticos voy a definir la información directamente en el “Store” utilizando un array de JavaScript, pero tu puedes sacar esta información de una base de datos y utilizar Ajax para cargar el Store, ya hemos estudiado esto anteriormente.

```
1. var store = new Ext.data.JsonStore({
2. fields: ['year', 'comedy', 'action', 'drama', 'thriller'],
3. data: [
4. {year: 2004, comedy: 39000000, action: 53890000, drama: 38450000, thriller: 32060000},
5. {year: 2005, comedy: 34000000, action: 23890000, drama: 18450000, thriller: 20060000},
6. {year: 2006, comedy: 56703000, action: 38900000, drama: 12650000, thriller: 21000000},
7. {year: 2007, comedy: 42100000, action: 50410000, drama: 25780000, thriller: 23040000},
8. {year: 2008, comedy: 38910000, action: 56070000, drama: 24810000, thriller: 26940000}
9.]
}]
```

```
10. });
```

Hemos definido algunos “géneros” de películas que contienen una cantidad de dinero que se generó en años anteriores, esta información es suficiente para poder realizar la gráfica.

### Creación de la gráfica

Vamos a crear la gráfica de columnas y usaremos una ventana para desplegarla en la pantalla, primero solamente graficaremos el género “comedy”.

```
1. var chart = new Ext.chart.ColumnChart({
2. store: store, // Step 1
3. xField: 'year', //Step 2
4. yField: 'comedy' //Step 3
5. });
```

En el paso uno le hemos asignado el “store”, este paso es muy importante pues es aquí donde se relaciona la información que necesitamos graficar.

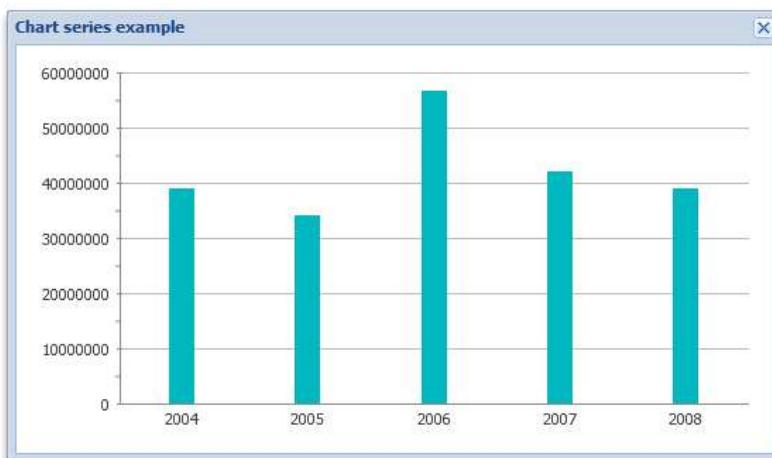
En el paso dos hemos definido el campo en el store (que asignamos en el paso uno) que será utilizado como el eje “X”.

En el paso tres definimos el campo donde saldrá la información para el eje “Y”.

Ahora necesitamos crear un contendor para renderizar la gráfica, en este caso utilizaremos una ventana con la siguiente configuración:

```
1. var win = new Ext.Window({
2. title: 'Chart series example',
3. width:550,
4. height:320,
5. layout:'fit',
6. items: chart
7. });
8.
9. win.show();
```

Esto es suficiente para que podamos ver un avance, al actualizar el explorador donde se encuentra nuestro ejemplo veremos algo como la siguiente imagen.



Gráfica muy sencilla

### Agregando series

Hasta ahora hemos graficado solamente una categoría, para poder incluir otras más dentro del mismo gráfico necesitamos usar la propiedad “series” y eliminar la propiedad “yField” puesto que será asignada por cada serie en particular.

```

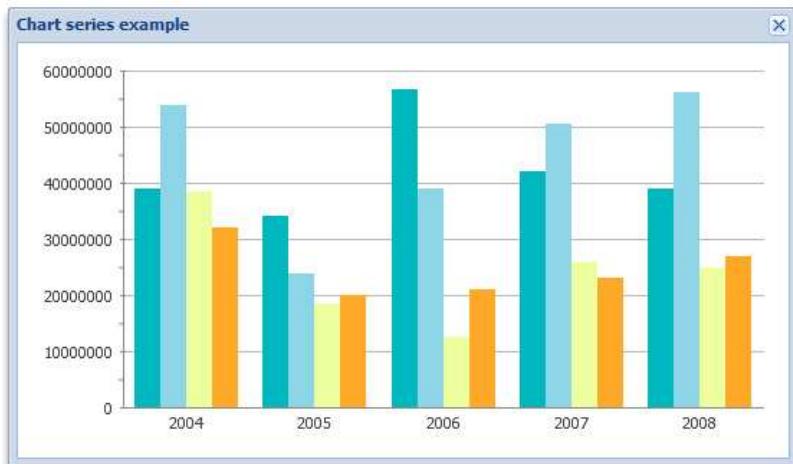
1. var chart = new Ext.chart.ColumnChart({
2. store: store,
3. xField: 'year',
4. //yField: 'comedy' //Step 1
5. series:[
6. {yField:'comedy'}, //Step 2
7. {yField:'action'},
8. {yField:'drama'},
9. {yField:'thriller'}
10.],
11. });

```

En el paso uno eliminamos la propiedad “yField” para poder asignarla mediante las series que definiremos más adelante.

En el paso dos hemos creado un arreglo con las series que usaremos, dentro de este arreglo irá la configuración de cada serie, es importante mencionar que podemos definir propiedades diferentes en cada serie.

El último paso, el tres, es la configuración de cada serie, hice la configuración lo más sencilla posible para una mejor comprensión, únicamente se define la propiedad “yField” asignándole el campo de donde será tomado el valor.



Desplegando las Series

Como pueden ver es extremadamente sencillo generar una gráfica que nos ayudará a visualizar de una manera fácil la comparación entre diferentes categorías, en este ejemplo usé géneros de películas, pero podemos utilizar cualquier otro tipo de información.

### Agregar leyenda

Para mejorar un poco nuestro gráfico es conveniente agregar una leyenda con el significado de cada barra y su color, esto lo hacemos de la siguiente manera.

```

1. var chart = new Ext.chart.ColumnChart({
2. store: store,
3. xField: 'year',
4. series:[
5. {yField:'comedy'},
6. {yField:'action'},
7. {yField:'drama'},
8. {yField:'thriller'}
9.],
10. extraStyle:{ //Step 1

```

```

11. legend:{ //Step 2
12. display: 'bottom'//Step 3
13. }
14. }
15. });

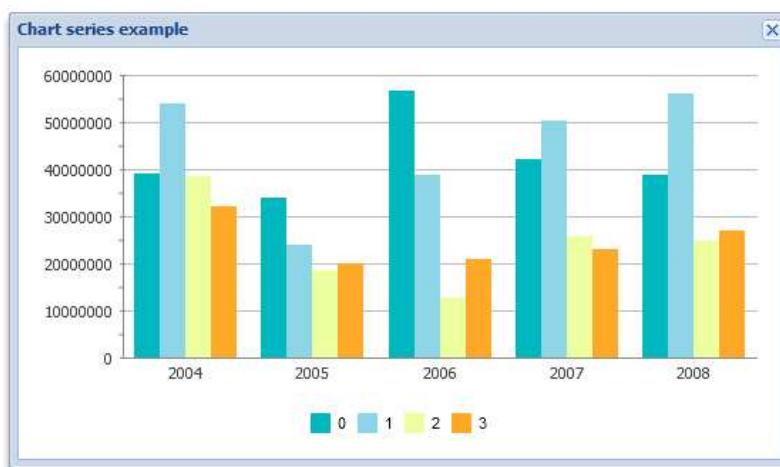
```

En el paso uno se definen estilos extra que serán utilizados por el gráfico, las propiedades definidas en este objeto sobre escribirán a las propiedades por defecto.

En el paso dos indicamos que agregaremos estilos a la leyenda.

En el paso tres únicamente le indicamos que queremos desplegar la leyenda en la parte inferior del gráfico, también podemos posicionarla al "top", "right", "left" o "bottom".

Para ver los cambios solamente es necesario refrescar la pantalla y veremos algo semejante a la siguiente imagen.



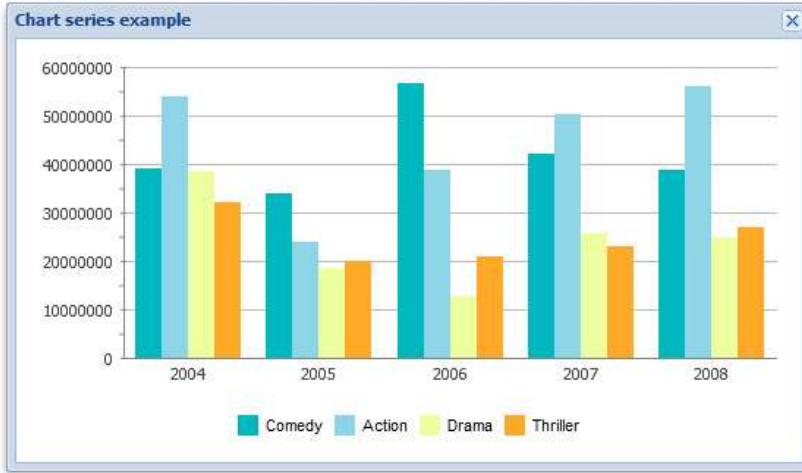
Mostrando una leyenda en la parte inferior del gráfico

### Cambiar el texto de la leyenda

Si eres observador habrás notado que el texto de la leyenda solamente son números, esto no le indica nada al usuario y además de que no es nada usable, necesitamos asignarle un texto más descriptivo.

1. {yField:'comedy',displayName:'Comedy'},
2. {yField:'action',displayName:'Action'},
3. {yField:'drama',displayName:'Drama'},
4. {yField:'thriller',displayName:'Thriller'}

Dentro de cada serie se ha configurado la propiedad "displayName" la cual nos permite asignarle un texto más amigable a la leyenda que desplegaremos.



Agregar textos más amigables a las etiquetas

### Formato a los ejes

Ahora vamos a ponerle formato al texto que se despliega en los ejes, en el eje "Y" necesitamos darle formato de moneda, para eso solamente agregamos la siguiente configuración dentro de la definición del ColumnChart.

1. yAxis: new Ext.chart.NumericAxis({
2.     labelRenderer: Ext.util.Format.usMoney
3. })

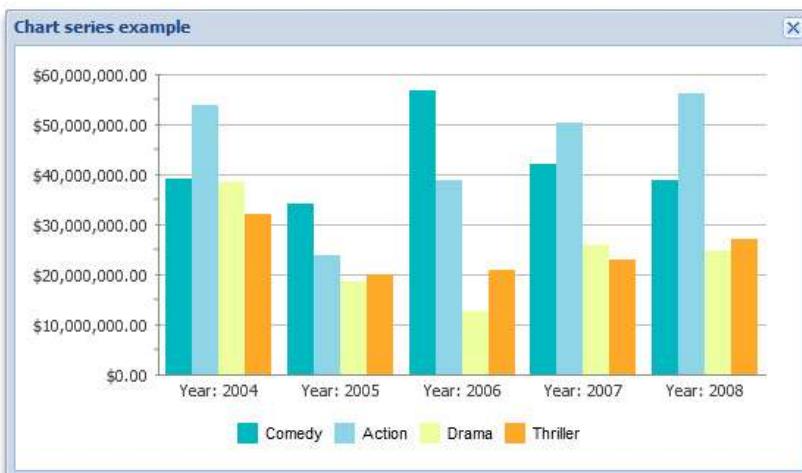
La propiedad "labelRenderer" acepta una función que procesará el contenido a desplegar por cada ítem del eje, esto es muy semejante al funcionamiento del "renderer" que usamos en los grids.

Ahora modificaremos el eje "X" asignando una función personalizada.

1. xAxis: new Ext.chart.CategoryAxis({
2.     labelRenderer: this.customFormat
3. })

Luego definimos la función "customFormat" que procesará el contenido a mostrar, esta función debe pertenecer al objeto principal.

1. customFormat:function(value){
2.     return 'Year: '+value;
3. }



Nótese que para los ejes se han usado dos componentes diferentes, “Ext.chart.NumericAxis” para crear un rango numérico y “Ext.chart.CategoryAxis” para utilizar las categorías que definimos en el store.

### **Conclusiones**

Es muy sencillo realizar gráficas con ExtJS, además el componente es muy flexible ya que nos permite personalizarlo fácilmente, he tratado de hacer el tutorial lo más sencillo posible, utilizando solamente las configuraciones básicas para el correcto funcionamiento pero te recomiendo que comiences a experimentar y jugar un rato con este componente.

## Las Tablas

Este componente es muy utilizado para desplegar información, permite realizar muchas tareas de manera fácil y rápida.

### Una tabla básica con información cargada de un Array

Uno de los controles más interesantes de ExtJS son las grillas, las cuales nos brindan la capacidad de poder mostrar la información de un modo sencillo y ordenado.

En este tutorial vamos a aprender a usar los componentes de una grilla básica como los DataStore o repositorios de datos, los ColumnModel, y los SelectionModel.

#### Empaquetando el tutorial

Vamos a empaquetar el código para evitar conflictos con otras variables.

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. Ext.BLANK_IMAGE_URL = './ext-3.0/resources/images/default/s.gif';
4.
5. com.quizzpot.tutorial.ArrayGridTutorial = {
6. init: function(){
7. //Aquí va el código del tutorial
8. }
9. }
10.
11. Ext.onReady(com.quizzpot.tutorial.ArrayGridTutorial.init,com.quizzpot.tutorial.ArrayGridTutorial);
```

#### El Store

Lo siguiente sería definir el repositorio de datos, es decir de donde van a ser obtenidos los datos, para este caso vamos a trabajar con datos estáticos obtenidos desde un Array bidimensional, de este modo:

```
1. //Arreglo bidimensional de datos
2. var myData = [
3. ['3m Co',71.72,0.02,0.03,'9/1 12:00am'],
4. ['Alcoa Inc',29.01,0.42,1.47,'9/1 12:00am'],
5. ['Altria Group Inc',83.81,0.28,0.34,'9/1 12:00am'],
6. ['American Express Company',52.55,0.01,0.02,'9/1 12:00am'],
7. ['American International Group, Inc.',64.13,0.31,0.49,'9/1 12:00am'],
8. ['AT&T Inc.',31.61,-0.48,-1.54,'9/1 12:00am'],
9. ['Boeing Co.',75.43,0.53,0.71,'9/1 12:00am'],
10. ['Caterpillar Inc.',67.27,0.92,1.39,'9/1 12:00am']
11.];
```

Ahora, vamos a definir un “Ext.data.ArrayStore” que va a encargarse de leer el arreglo de datos, para esto debemos indicarle cual va a ser el nombre de referencia de la columna de datos y el tipo de dato que contiene.

```
1. //creando el repositorio de datos
2. var store = new Ext.data.ArrayStore({
3. fields: [
4. {name: 'compania'},
5. {name: 'precio', type: 'float'},
6. {name: 'cambio', type: 'float'},
7. {name: 'pctCambio', type: 'float'},
8. {name: 'actualizado', type: 'date', dateFormat: 'n/j h:ia'}
9.]
10. });
11. store.loadData(myData);
```

El atributo “name”, define el nombre con el que vamos a referenciar la primera columna de datos, la columna llamada “price” será la referencia a la segunda columna y así sucesivamente.

Se pueden emplear una gran cantidad de atributos para las referencias a las columnas, como “type” que define el tipo de dato que alberga la columna, “dateFormat” define el formato de las columnas de tipo date(formato ExtJS ver API).

## La Tabla

Ahora estamos listos para crear nuestra tabla “instanciando” al objeto “Ext.grid.GridPanel”:

```
1. //Creando el objeto Ext.grid.GridPanel
2. var grid = new Ext.grid.GridPanel({
3. title:'Listado de Compañías',
4. store: store,
5. renderTo: document.body,
6. columns: [
7. {id:'compania',header: "Compañía", width: 160, sortable: true, dataIndex: 'compania'},
8. {header: "Precio", width: 75, sortable: true, dataIndex: 'precio'},
9. {header: "Cambio", width: 75, sortable: true, dataIndex: 'cambio'},
10. {header: "% de cambio", width: 75, sortable: true, dataIndex: 'pctCambio'},
11. {header: "Actualizado", width: 85, sortable: true, renderer: Ext.util.Format.dateRenderer('m/d/Y'), dataIndex: 'actualizado'}
12.],
13. stripeRows: true,
14. height:250,
15. width:500
16. });


```

Las tablas al igual que los formularios y otros componentes heredan de “Ext.Panel”, esto ya ha sido visto en capítulos anteriores así que imagino que estarán familiarizados con esto.

Las Propiedades usadas:

“title”: propiedad heredada de la clase Ext.Panel, define el título del panel.

“store”: propiedad que define de donde la tabla va a obtener los datos, (el Ext.data.ArrayStore definido anteriormente).

“renderTo”: define donde la va “renderizarse” (pintarse) la tabla, recibe una referencia al id del un objeto DOM como un DIV.

“columns”: define el encabezado de cada columna, aquí se relacionan los datos del store por medio de la propiedad “name” definida en el store.

“id”: define el id de la columna.

“header”: define el nombre a mostrar en el título de cada columna.

“width”: define el ancho de la columna.

“sortable”: propiedad “booleanas” que define si se va a permitir el ordenamiento de la data al darle clic al título de la columna.

“dataIndex”: aquí se indica el valor de la propiedad “name” del store, con esto asociamos la data de la columna del store con la columna de la tabla.

“renderer”: con esta propiedad podemos personalizar como se muestran los datos de la columna, lo explicaré más adelante.

“stripeRows”: sirve para mostrar un ligero sombreado interlineado en las filas de la tabla.

“height”: define el alto de la tabla.

“width”: define el ancho de la tabla.

| Listado de Compañías             |        |        |             |             |  |
|----------------------------------|--------|--------|-------------|-------------|--|
| Compañía                         | Precio | Cambio | % de cambio | Actualizado |  |
| 3m Co                            | 71.72  | 0.02   | 0.03        | 09/01/2009  |  |
| Alcoa Inc                        | 29.01  | 0.42   | 1.47        | 09/01/2009  |  |
| Altria Group Inc                 | 83.81  | 0.28   | 0.34        | 09/01/2009  |  |
| American Express Company         | 52.55  | 0.01   | 0.02        | 09/01/2009  |  |
| American International Group,... | 64.13  | 0.31   | 0.49        | 09/01/2009  |  |
| AT&T Inc.                        | 31.61  | -0.48  | -1.54       | 09/01/2009  |  |
| Boeing Co.                       | 75.43  | 0.53   | 0.71        | 09/01/2009  |  |
| Caterpillar Inc.                 | 67.27  | 0.92   | 1.39        | 09/01/2009  |  |

Imagen de una tabla básica

Como podemos ver las tablas tienen muy bien definidos sus componentes, para el manejo de datos se usa un store cuyo tipo dependerá del tipo de información que maneje por ejemplo existe el “Ext.data.JSONStore” preparado para poder manejar información en formato JSON, sugiero revisar el API para poder ver más propiedades y tipos de repositorios de datos.

### El ColumnModel

Si bien nuestra grilla ahora está lista para usarse y hay algunas cosas que podríamos necesitar que afortunadamente ExtJS trae consigo; una de esas cosas son los “Ext.grid.ColumnModel”, los cuales sirven para poder presentar las columnas de la tabla y relacionar las mismas con las columnas del store, pero a su vez nos proveen funcionalidades de por ejemplo, de selección de filas mediante un checkbox o una columna para la enumeración de las filas.

Básicamente la creación de un columnModel es similar al contenido que hemos agregado en la propiedad “columns” de nuestra tabla.

```
1. //Creando el objeto Ext.grid.ColumnModel
2. var myColumnModel = new Ext.grid.ColumnModel([
3. {id:'compania',header: "Compañía", width: 160, sortable: true, dataIndex: 'compania'},
4. {header: "Precio", width: 75, sortable: true, dataIndex: 'precio'},
5. {header: "Cambio", width: 75, sortable: true, dataIndex: 'cambio'},
6. {header: "% de cambio", width: 75, sortable: true, dataIndex: 'pctCambio'},
7. {header: "Actualizado", width: 85, sortable: true, renderer: Ext.util.Format.dateRenderer('m/d/Y')
 , dataIndex: 'actualizado'}
8.]);
9. //nuestra grilla cambiaria para referenciar a al variable myColumnModel
10. var grid = new Ext.grid.GridPanel({
11. title:'Listado de Compañías',
12. store: store,
13. renderTo: document.body,
14. cm: myColumnModel,//referencia al columnModel
15. stripeRows: true,
16. height:250,
17. width:500
18. });
```

Ahora vamos a agregar una numeración para nuestras filas así como una columna de checkboxes para selección.

```

1. //Creando el objeto Ext.grid.ColumnModel
2. var myColumnModel = new Ext.grid.ColumnModel([
3. new Ext.grid.RowNumberer(),
4. new Ext.grid.CheckboxSelectionModel(),
5. {id:'compania',header: "Compañía", width: 160, sortable: true, dataIndex: 'compania'},
6. ...
7.]);

```

| Listado de Compañías |                                  |        |        |             |             |
|----------------------|----------------------------------|--------|--------|-------------|-------------|
|                      | Compañía                         | Precio | Cambio | % de cambio | Actualizado |
| 1                    | 3m Co                            | 71.72  | 0.02   | 0.03        | 09/01/2009  |
| 2                    | Alcoa Inc                        | 29.01  | 0.42   | 1.47        | 09/01/2009  |
| 3                    | Altria Group Inc                 | 83.81  | 0.28   | 0.34        | 09/01/2009  |
| 4                    | American Express Company         | 52.55  | 0.01   | 0.02        | 09/01/2009  |
| 5                    | American International Group,... | 64.13  | 0.31   | 0.49        | 09/01/2009  |
| 6                    | AT&T Inc.                        | 31.61  | -0.48  | -1.54       | 09/01/2009  |
| 7                    | Boeing Co.                       | 75.43  | 0.53   | 0.71        | 09/01/2009  |
| 8                    | Caterpillar Inc.                 | 67.27  | 0.92   | 1.39        | 09/01/2009  |

Una tabla con numeración en filas y con checkbox para ser seleccionadas

### El SelectionModel

Finalmente solo nos queda agregar un “Ext.grid.SelectionModel” el cual el va a indicar el modo de selección que va a tener nuestra tabla.

```

1. //creamos una variable para referenciar a nuestro objeto
2. var mySelectionModel = new Ext.grid.CheckboxSelectionModel({singleSelect: false});
3.
4. //nuestra grilla quedaria finalmente asi
5. var grid = new Ext.grid.GridPanel({
6. title:'Listado de Compañías',
7. store: store,
8. renderTo: document.body,
9. cm: myColumnModel, //referencia al columnModel
10. sm: mySelectionModel, //referencia al selectionModel
11. stripeRows: true,
12. height:250,
13. width:500
14. });

```

| Listado de Compañías |                                  |        |        |             |             |
|----------------------|----------------------------------|--------|--------|-------------|-------------|
|                      | Compañía                         | Precio | Cambio | % de cambio | Actualizado |
| 1                    | 3m Co                            | 71.72  | 0.02   | 0.03        | 09/01/2009  |
| 2                    | Alcoa Inc                        | 29.01  | 0.42   | 1.47        | 09/01/2009  |
| 3                    | Altria Group Inc                 | 83.81  | 0.28   | 0.34        | 09/01/2009  |
| 4                    | American Express Company         | 52.55  | 0.01   | 0.02        | 09/01/2009  |
| 5                    | American International Group,... | 64.13  | 0.31   | 0.49        | 09/01/2009  |
| 6                    | AT&T Inc.                        | 31.61  | -0.48  | -1.54       | 09/01/2009  |
| 7                    | Boeing Co.                       | 75.43  | 0.53   | 0.71        | 09/01/2009  |
| 8                    | Caterpillar Inc.                 | 67.27  | 0.92   | 1.39        | 09/01/2009  |

Características de una tabla en Ext JS

## Conclusiones

Hoy hemos creado una tabla simple que lee datos de un Array por medio de un "ArrayStore", luego se crea la tabla y se asocian las columnas del store por medio de la propiedad "columns" de la tabla o creando un "ColumnModel" y finalmente agregamos un "SelectionModel" para gestionar la forma de selección de las filas, próximamente veremos como usar diferentes tipos de store, usar eventos en el store, así como la tabla y el "seleccionModel".

## Mostrar información de un archivo XML

El día de hoy voy a mostrar como cargar información en un grid a partir de un archivo XML, esto es muy sencillo de realizar y útil a la hora de desarrollar aplicaciones.

### Material de apoyo

Para continuar necesitamos descargar el material de apoyo, y copiar los tres archivos dentro de nuestro servidor Web, hemos estado trabajando dentro de la carpeta "curso" donde ya tenemos la librería de Ext JS y además ya creamos una carpeta que se llama "grids" para este capítulo.

Antes de continuar puedes ver la demostración de lo que vamos a estar haciendo en este tutorial.



|   | Name          | Company             | Position                             | Age |
|---|---------------|---------------------|--------------------------------------|-----|
| 1 | Jack Slocum   | Ext JS              | Chief Software Architect and Founder | 32  |
| 2 | Sasha Cohen   |                     | Figure Skating                       | 24  |
| 3 | John Resig    | Mozilla Corporation | JavaScript Developer                 | 24  |
| 4 | Sara Mcfly    | Google              | Tester                               | 35  |
| 5 | Crysfel Villa | JVM Solutions       | Software Developer                   | 25  |
| 6 | Felicia Day   |                     | Actress                              | 30  |
| 7 | Collis T'aeed | Envato              | CEO                                  | 29  |

Ejemplo final

### Empaquetando el código

Ya sabemos que empaquetar nuestro código es una muy buena práctica, así que vamos a crear el namespace o "paquete" para este tutorial.

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.GridXmlTutorial = {
4. init: function(){
5. //code goes here
6. }
7. }
8.
9. Ext.onReady(com.quizzpot.tutorial.GridXmlTutorial.init,com.quizzpot.tutorial.GridXmlTutorial);
```

### El XML a utilizar

Vamos a utilizar XML como nuestra fuente de datos, estos datos pueden estar contenidos en una base de datos, un servicio Web, un archivo, etc. Para hacer las cosas más sencillas en este ejemplo he puesto la información directamente en el código fuente de la siguiente manera:

```
1. <?php
2. header("Content-Type: text/xml");
3.
4. echo '<?xml version="1.0" encoding="UTF-8"?>';
5. ?
6.
7. <people>
```

```

8. <person>
9. <name>Jack Slocum</name>
10. <age>32</age>
11. <position>Chief Software Architect and Founder</position>
12. <company>Ext JS</company>
13. </person>
14. <person>
15. <name>Sasha Cohen</name>
16. <age>24</age>
17. <position>Figure Skating</position>
18. <company></company>
19. </person>
20. <person>
21. <name>John Resig</name>
22. <age>24</age>
23. <position>JavaScript Developer</position>
24. <company>Mozilla Corporation</company>
25. </person>
26. <person>
27. <name>Sara Mcfly</name>
28. <age>35</age>
29. <position>Tester</position>
30. <company>Google</company>
31. </person>
32. <person>
33. <name>Crysfel Villa</name>
34. <age>25</age>
35. <position>Software Developer</position>
36. <company>JWM Solutions</company>
37. </person>
38. <person>
39. <name>Felicia Day</name>
40. <age>30</age>
41. <position>Actress</position>
42. <company></company>
43. </person>
44. <person>
45. <name>Collis Ta'eed</name>
46. <age>29</age>
47. <position>CEO</position>
48. <company>Envato</company>
49. </person>
50. </people>

```

### **El registro “Person”**

Vamos a desplegar en el grid la información contenida en el XML, estos son datos de personas, por lo tanto vamos a crear el registro “Person” y ‘mapearlo’ con el nodo “person” del XML.

```

1. var Person = Ext.data.Record.create([
2. {name: 'name'},
3. {name: 'position'},
4. {name: 'age', type:'float'},
5. {name: 'company'}
6.]);

```

### **Crear el “Reader”**

Ahora vamos a crear el “lector” que le asignaremos al “store”:

```

1. var reader = new Ext.data.XmlReader({
2. record: "person"

```

```
3. }, Person);
```

```
4.
```

Aquí solamente hemos definido en la configuración del “reader” la propiedad “record” y le hemos asignado el valor “person”, con esto el “reader” será capaz de ir a los nodos “person” (en el XML) y extraer la información contenida en el objeto “Person” que definimos anteriormente.

### Crear el Store y cargar la información

El siguiente paso es crear el “Store” y cargar la información del XML, esto se hace de la siguiente manera:

```
1. var store = new Ext.data.Store({
2. url: 'xml.php',
3. reader: reader
4. });
5.
6. store.load();
```

Primero se define la “url” donde se va a buscar el XML mediante Ajax, el segundo parámetro de configuración es asignarle el “reader” al “Store”.

Una vez creado el store podemos utilizar el método “load” para realizar la petición “Ajax” y recibir la información.

Hasta aquí solamente hemos definido la fuente de datos, haciendo las configuraciones necesarias. Si no te ha quedado clara esta parte del “Store”, te recomiendo estudiar el tema donde hablamos específicamente de este componente.

### Crear el Grid

Una vez que tenemos lista la información a desplegar en la tabla ya podemos comenzar a crear el grid de la siguiente manera:

```
1. var grid = new Ext.grid.GridPanel({
2. store: store, //le asignamos la fuente de datos
3. columns: [//creamos las columnas
4. new Ext.grid.RowNumberer(), //numeramos las filas
5. {header:'Name', dataIndex:'name', sortable: true},
6. {header:'Company', dataIndex:'company', sortable: true},
7. {header:'Position', dataIndex:'position', width:230, sortable: true},
8. {header:'Age', dataIndex:'age', width:40, sortable: true}
9.],
10. border: false, //le quitamos el borde
11. stripeRows: true //le asignamos rayas a las filas
12.});
```

Con esa sencilla configuración creamos una tabla muy básica pero atractiva, ya hemos estudiado este código en el tema anterior, pero a manera de repaso voy a mencionar las propiedades utilizadas:

“store”: En esta propiedad se define la fuente de datos a utilizar en el grid.

“columns”: Se definen las columnas que tendrá el grid, esta parte es muy importante pues aquí se relacionan las columnas y su información a mostrar (dataIndex).

“border”: Esta propiedad se hereda del componente Panel y al asignarle “false” le quitamos el borde exterior, esto lo hacemos porque vamos a meter el grid en una ventana.

“stripeRows”: Nos permite ver unas rayas en las filas, esto hace que sea más fácil visualizar la información.

### Crear la ventana

Por último vamos a meter el grid dentro de una ventana:

```

1. var win = new Ext.Window({
2. title: 'Grid example',
3. layout: 'fit', // <---
4. width: 510,
5. height:350,
6. items: grid
7. });
8.
9. win.show();

```

Lo único importante en esta configuración y que probablemente no sabemos (si hemos tomado este curso desde el principio) es la configuración “layout” a la cual le estamos asignando “fit”, esto es para que la tabla ocupe el cien por ciento de la ventana, de lo contrario el grid no se desplegará completamente, más adelante voy a tratar el tema de los layouts.

|   | Name          | Company             | Position                             | Age |
|---|---------------|---------------------|--------------------------------------|-----|
| 1 | Jack Slocum   | Ext JS              | Chief Software Architect and Founder | 32  |
| 2 | Sasha Cohen   |                     | Figure Skating                       | 24  |
| 3 | John Resig    | Mozilla Corporation | JavaScript Developer                 | 24  |
| 4 | Sara Mcfly    | Google              | Tester                               | 35  |
| 5 | Crystel Villa | J/M Solutions       | Software Developer                   | 25  |
| 6 | Felicia Day   |                     | Actress                              | 30  |
| 7 | Collis Ta'eed | Envato              | CEO                                  | 29  |

Si todo ha salido bien, veras esta pantalla

### Conclusiones

Hemos visto lo sencillo que es desplegar información de un XML en un grid, además hemos apreciado la ingeniería con que está realizada la librería de Ext JS, porque la creación del grid es la misma para los diferentes formatos (XML, JSON, Array), lo único que hemos cambiado ha sido el “Store” y el grid funciona igual, esto es una ventaja en cuanto a el mantenimiento de sistemas se refiere.

## Información contenida en formato JSON

El día de hoy quiero mostrar como cargar información contenida en formato JSON en un “Grid”, es realmente sencillo y básico, pero creo que es importante mencionarlo para aquellos que están empezando a trabajar con esta librería.

Este tutorial es muy semejante al anterior donde vimos como cargar información desde un archivo XML, lo único que cambiaremos con respecto al tutorial anterior es el registro donde se especifica la información que tendrá y el “reader” para que sea capaz de leer el JSON regresado por el servidor.

|    | City          | Visits | Page/Visits | Average Time |
|----|---------------|--------|-------------|--------------|
| 1  | Mexico city   | 684    | 4.11        | 00:06:53     |
| 2  | La Victoria   | 443    | 4.39        | 00:07:28     |
| 3  | Madrid        | 380    | 3.11        | 00:05:22     |
| 4  | Providencia   | 204    | 3.83        | 00:08:20     |
| 5  | Bogota        | 204    | 3.26        | 00:04:57     |
| 6  | Puerto Madero | 192    | 3.56        | 00:05:07     |
| 7  | Monterrey     | 174    | 3.9         | 00:06:06     |
| 8  | Barcelona     | 145    | 3.28        | 00:05:39     |
| 9  | Caracas       | 132    | 4.55        | 00:06:27     |
| 10 | Rosario       | 116    | 2.44        | 00:04:30     |
| 11 | Oaxaca        | 108    | 1.73        | 00:02:37     |

Imagen final

### Material de apoyo

Vamos a descargar el material de apoyo, lo descomprimimos y copiamos dentro de la carpeta “grid” en nuestro servidor Web previamente instalado.

### Empaquetando el tutorial

Como siempre hemos hecho en este curso, vamos a “empaquetar” el código que usaremos, recuerden que esto es fundamental y muy importante, realmente nunca me cansaré de decir esto.

1. Ext.ns('com.quizzpot.tutorial');
- 2.
3. com.quizzpot.tutorial.GridJsonTutorial = {
4.     init: function(){
5.         //code goes here
6.     }
- 7.
8. Ext.onReady(com.quizzpot.tutorial.GridJsonTutorial.init,com.quizzpot.tutorial.GridJsonTutorial);

## El JSON a utilizar

Ahora vamos a definir la información que desplegará el grid, por cuestiones de simplicidad voy a escribir la información directamente en el código, pero les recuerdo que esta información puede estar en una base de datos, un archivo de texto, un servicio Web (Web Service) o provenir de cualquier otro lugar, por ahora está “hardcoded”.

```
1. <?php
2. header("Content-Type: text/plain");
3.
4. $data = array(
5. 'success'=>true,
6. 'total'=>11,
7. 'data'=>array(
8. array('city'=>'Mexico city','visits'=>684,'pageVisits'=>4.11,'averageTime'=>'00:06:53'),
9. array('city'=>'La Victoria','visits'=>443,'pageVisits'=>4.39,'averageTime'=>'00:07:28'),
10. array('city'=>'Madrid','visits'=>380,'pageVisits'=>3.11,'averageTime'=>'00:05:22'),
11. array('city'=>'Providencia','visits'=>204,'pageVisits'=>3.83,'averageTime'=>'00:08:20'),
12. array('city'=>'Bogota','visits'=>204,'pageVisits'=>3.26,'averageTime'=>'00:04:57'),
13. array('city'=>'Puerto Madero','visits'=>192,'pageVisits'=>3.56,'averageTime'=>'00:05:07'),
14. array('city'=>'Monterrey','visits'=>174,'pageVisits'=>3.90,'averageTime'=>'00:06:06'),
15. array('city'=>'Barcelona','visits'=>145,'pageVisits'=>3.28,'averageTime'=>'00:05:39'),
16. array('city'=>'Caracas','visits'=>132,'pageVisits'=>4.55,'averageTime'=>'00:06:27'),
17. array('city'=>'Rosario','visits'=>116,'pageVisits'=>2.44,'averageTime'=>'00:04:30'),
18. array('city'=>'Oaxaca','visits'=>108,'pageVisits'=>1.73,'averageTime'=>'00:02:37')
19.)
20.);
21.
22. echo json_encode($data);
23. ?>
```

## Crear el “Record”

Una vez definido el formato con el cual será entregada la información al cliente (por medio de AJAX), podemos crear el “Record” que desplegaremos en la tabla.

```
1. var Record = Ext.data.Record.create([
2. {name: 'city'},
3. {name: 'visits', type:'float'},
4. {name: 'pageVisits', type:'float'},
5. {name: 'averageTime'}
6.]);
```

El código anterior ya debe ser conocido por nosotros, pues lo hemos repasado en temas anteriores, básicamente creamos un “registro” con los campos que necesitamos desplegar y que son entregados por el servidor, además definimos (en algunos casos) el tipo de información del campo.

## Crear el “Reader”

Ahora vamos a escribir el “Reader” para que el store pueda interpretar la información que se nos entrega en formato JSON.

```
1. var reader = new Ext.data.JsonReader({
2. totalRecords: "total",
3. root: "data"
4. }, Record);
```

Para este caso utilizamos el componente “JsonReader” y le configuramos el campo donde se encuentra el total de registros y el “root” donde viene la información principal.

## **Crear el Store y cargar la información**

Lo siguiente es crear el store que se encargará de contener localmente la información para poder ser manipulada por el grid.

```
1. var store = new Ext.data.Store({
2. url: 'gridjson.php',
3. reader: reader
4. });
5.
6. store.load();
```

Puedes ver que se ha configurado la “url” donde solicitará por medio de Ajax la información a contener, además le hemos asignado el “Reader” que creamos anteriormente, por último hacemos un “load”.

## **Ahormando algunas líneas de código**

Hasta ahora lo único que hemos hecho es ir por la información al servidor y contenerla dentro del store, todas las líneas anteriores de código pueden ser reducidas considerablemente de la siguiente manera:

```
1. var store = new Ext.data.JsonStore({
2. url: 'gridjson.php',
3. root: 'data',
4. fields: ['city',{name:'visits',type:'float'},{name:'pageVisits',type:'float'},'averageTime']
5. });
6.
7. store.load();
```

Con las líneas anteriores hacemos exactamente lo mismo que hicimos anteriormente, esto es conveniente cuando utilizamos JSON como formato de transferencia de información, así que la decisión de realizarlo de una u otra manera queda a consideración del desarrollador.

## **Crear el Grid**

Vamos a crear el grid que desplegará la información contenida en el store que creamos anteriormente.

```
1. var grid = new Ext.grid.GridPanel({
2. store: store, // <--- le asignamos el store con la información a utilizar
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'City', dataIndex:'city', sortable: true},
6. {header:'Visits', dataIndex:'visits', sortable: true},
7. {header:'Page/Visits', dataIndex:'pageVisits', sortable: true},
8. {header:'Average Time', dataIndex:'averageTime', width:150, sortable: true}
9.],
10. border: false,
11. stripeRows: true
12.});
```

Aquí no estamos haciendo nada en especial, solamente le asignamos el store que utilizará, definimos las columnas y a cada una le asignamos una propiedad del “Record” en el store, además creamos una columna donde se numeran las filas, le quitamos el borde para que al introducir la tabla a una ventana se vea bien, y le agregamos líneas alternadas en las filas.

## **Desplegar el grid**

Existen varias alternativas para desplegar el grid en la pantalla, en esta ocasión vamos a utilizar una ventana.

```
1. var win = new Ext.Window({
2. title: 'Grid example',
3. layout: 'fit',
4. width: 510,
```

```
5. height:350,
6. items: grid
7. });
8.
9. win.show();
```

Si actualizamos el explorador veremos algo como la siguiente imagen.

|    | City          | Visits | Page/Visits | Average Time |
|----|---------------|--------|-------------|--------------|
| 1  | Mexico city   | 684    | 4.11        | 00:06:53     |
| 2  | La Victoria   | 443    | 4.39        | 00:07:28     |
| 3  | Madrid        | 380    | 3.11        | 00:05:22     |
| 4  | Providencia   | 204    | 3.83        | 00:08:20     |
| 5  | Bogota        | 204    | 3.26        | 00:04:57     |
| 6  | Puerto Madero | 192    | 3.56        | 00:05:07     |
| 7  | Monterrey     | 174    | 3.9         | 00:06:06     |
| 8  | Barcelona     | 145    | 3.28        | 00:05:39     |
| 9  | Caracas       | 132    | 4.55        | 00:06:27     |
| 10 | Rosario       | 116    | 2.44        | 00:04:30     |
| 11 | Oaxaca        | 108    | 1.73        | 00:02:37     |

Imagen final

## Conclusiones

Hasta ahora solamente hemos visto como desplegar la información en diferentes formatos, esto es realmente sencillo. En el siguiente capítulo veremos como paginar esta información, cada vez se pone mas interesante este curso por lo tanto no olviden inscribirse a [las Feeds](#) o [por email](#), además recuerden que utilizamos Twitter (quizzpot) para mostrar las actualizaciones y lo que estamos haciendo en Quizzpot.

## Paginado remoto en un Grid

El día de hoy voy a mostrar como podemos paginar los registros de una tabla, esta funcionalidad es muy utilizada cuando tenemos una gran cantidad de información para mostrar al usuario.

El ejercicio de hoy lo puedes ver en acción, a continuación te muestro una imagen de lo que tendremos al final del tutorial.

|   | City        | Visits | Page/Visits | Average Time |
|---|-------------|--------|-------------|--------------|
| 1 | Mexico city | 684    | 4.11        | 00:06:53     |
| 2 | La Victoria | 443    | 4.39        | 00:07:28     |
| 3 | Madrid      | 380    | 3.11        | 00:05:22     |
| 4 | Providencia | 204    | 3.83        | 00:08:20     |
| 5 | Bogota      | 204    | 3.26        | 00:04:57     |

Page 1 of 3 | 1 - 5 of 14 Cities

### Ejercicio final

#### Material de apoyo

Antes de continuar es necesario descargar el material de apoyo y copiar los archivos dentro de la carpeta "grids" que se encuentra en el servidor Web que instalamos en el primer capítulo y en el cual hemos estado trabajando.

#### Definir el "namespace"

Antes que cualquier otra cosa necesitamos definir el "namespace" donde estaremos trabajando.

```

1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.PagingTutorial = {
4. init: function(){
5. //code goes here
6. }
7. }
8.
9. Ext.onReady(com.quizzpot.tutorial.PagingTutorial.init,com.quizzpot.tutorial.PagingTutorial);

```

#### El paginado

Vamos a paginar la información en el servidor, éste recibirá dos parámetros, "start" y "limit", estos parámetros son enviados automáticamente por la barra paginadora.

```

1. <?php
2. header("Content-Type: text/plain");
3.
4. $start = isset($_POST['start'])?$_POST['start']:0; //posición a iniciar
5. $limit = isset($_POST['limit'])?$_POST['limit']:5; //número de registros a mostrar
6.
7. $data = array(
8. array('city'=>'Mexico city','visits'=>684,'pageVisits'=>4.11,'averageTime'=>'00:06:53'),
9. array('city'=>'La Victoria','visits'=>443,'pageVisits'=>4.39,'averageTime'=>'00:07:28'),
10. array('city'=>'Madrid','visits'=>380,'pageVisits'=>3.11,'averageTime'=>'00:05:22'),
11. array('city'=>'Providencia','visits'=>204,'pageVisits'=>3.83,'averageTime'=>'00:08:20'),
12. array('city'=>'Bogota','visits'=>204,'pageVisits'=>3.26,'averageTime'=>'00:04:57'),
13. array('city'=>'Puerto Madero','visits'=>192,'pageVisits'=>3.56,'averageTime'=>'00:05:07'),
14. array('city'=>'Monterrey','visits'=>174,'pageVisits'=>3.90,'averageTime'=>'00:06:06'),
15. array('city'=>'Barcelona','visits'=>145,'pageVisits'=>3.28,'averageTime'=>'00:05:39'),

```

```

16. array('city'=>'Caracas','visits'=>132,'pageVisits'=>4.55,'averageTime'=>'00:06:27'),
17. array('city'=>'Rosario','visits'=>116,'pageVisits'=>2.44,'averageTime'=>'00:04:30'),
18. array('city'=>'Oaxaca','visits'=>108,'pageVisits'=>1.73,'averageTime'=>'00:02:37'),
19. array('city'=>'Buenos Aires','visits'=>100,'pageVisits'=>5.43,'averageTime'=>'00:07:37'),
20. array('city'=>'Galicia','visits'=>96,'pageVisits'=>1.92,'averageTime'=>'00:04:37'),
21. array('city'=>'Guadalajara','visits'=>90,'pageVisits'=>5.92,'averageTime'=>'00:03:37')
22.);
23.
24. $paging = array(
25. 'success'=>true,
26. 'total'=>count($data), //<--- total de registros a paginar
27. 'data'=> array_splice($data,$start,$limit)
28.);
29.
30. echo json_encode($paging);
31. ?>

```

El código anterior viene con el material de apoyo, realmente no tiene nada especial únicamente recibe los parámetros y mediante la función “array\_splice” regresa los registros solicitados, es importante mencionar que esta información puede provenir de una base de datos o de cualquier otro lugar, pero lo que si debes tener en cuenta es que la paginación depende totalmente del servidor y éste es el responsable de regresar la información correcta.

### Crear el Store

El siguiente paso es crear un JsonStore para solicitar los registros por medio de Ajax de la siguiente manera:

```

1. var store = new Ext.data.JsonStore({
2. url: 'paging.php',
3. root: 'data',
4. totalProperty: 'total', // <--- total de registros a paginar
5. fields: ['city','visits','pageVisits','averageTime']
6. });
7.
8. store.load();

```

Si has venido siguiendo el curso desde el principio el código anterior debe ser familiar, la única propiedad que quiero resaltar es “*totalProperty*” ya que esta propiedad define donde se encuentra (en la información que nos regresa el servidor) el total de los elementos a paginar; si no configuramos la propiedad antes mencionada, el componente no podrá calcular el número de páginas porque supone que el total es solamente los registros que se están mostrando actualmente, por lo tanto no permitirá seguir paginando la información.

### Crear el Grid

Vamos a crear el Grid y asignárselo a una ventana para poder visualizarlo.

```

1. var grid = new Ext.grid.GridPanel({
2. store: store,
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'City', dataIndex:'city', sortable: true},
6. {header:'Visits', dataIndex:'visits', sortable: true},
7. {header:'Page/Visits', dataIndex:'pageVisits', sortable: true},
8. {header:'Average Time', dataIndex:'averageTime', width:150, sortable: true}
9.],
10. border: false,
11. stripeRows: true
12. });
13.

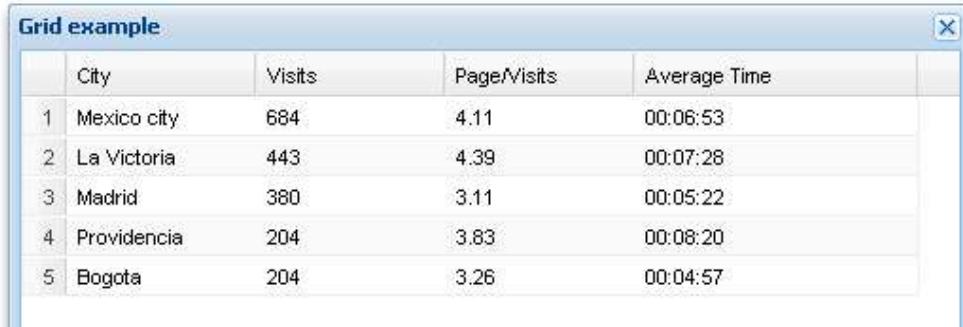
```

```

14. var win = new Ext.Window({
15. title: 'Grid example',
16. layout: 'fit',
17. width: 510,
18. height:350,
19. items: grid
20. });
21.
22. win.show();

```

El código anterior es muy semejante al del tutorial anterior, lo único que hace es crear la tabla con las columnas necesarias y relacionar las columnas con los campos en el registro del store.



|   | City        | Visits | Page/Visits | Average Time |
|---|-------------|--------|-------------|--------------|
| 1 | Mexico city | 684    | 4.11        | 00:06:53     |
| 2 | La Victoria | 443    | 4.39        | 00:07:28     |
| 3 | Madrid      | 380    | 3.11        | 00:05:22     |
| 4 | Providencia | 204    | 3.83        | 00:08:20     |
| 5 | Bogota      | 204    | 3.26        | 00:04:57     |

Creación de una tabla

### Crear el PagingToolbar

Vamos a crear la barra de paginación de la siguiente manera:

```

1. var pager = new Ext.PagingToolbar({
2. store: store, // <--grid and PagingToolbar using same store (required)
3. displayInfo: true,
4. displayMsg: '{0} - {1} of {2} Cities',
5. emptyMsg: 'No cities to display',
6. pageSize: 5
7. });

```

He configurado las propiedades más importantes, pero la única propiedad requerida es “store” el cual debe ser el mismo que se usa para la tabla.

“displayInfo” (boolean): Define si se despliega información en la parte derecha del componente sobre el estado de la paginación, por defecto es “false”.

“displayMsg” (string): Especifica el mensaje que se desea desplegar cuando la propiedad anterior (“displayInfo”) es seleccionada “true”, el mensaje por defecto es “Displaying {0} – {1} of {2}”, en este caso lo hemos cambiado, es importante notar que {0} será remplazado por el parámetro “start”, {1} se remplazará por “star + limit” y {2} se remplazará por “totalProperty”.

“emptyMsg” (string): Este mensaje se mostrará cuando el servidor no regrese ningún registro, si no la configuramos mostrará “No data to display” por defecto.

“pageSize” (number) : Aquí definimos el número de registros por página, en este caso le asignamos cinco, pero queda a criterio del desarrollador, solamente se debe tener en cuenta que a mayor registros por página más difícil será para el usuario encontrar lo que necesita, por defecto la configuración de esta propiedad es 20.

### Asignar la barra de paginación al Grid

Una vez que hemos creado nuestra barra de paginación se la podemos asignar al grid en la barra superior (tbar) o inferior (bbar), en este caso se la voy asignar en la parte inferior de la siguiente manera:

```

1. var grid = new Ext.grid.GridPanel({
2. store: store, // <--El grid y PagingToolbar usan el mismo store
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'City', dataIndex:'city', sortable: true},
6. {header:'Visits', dataIndex:'visits', sortable: true},
7. {header:'Page/Visits', dataIndex:'pageVisits', sortable: true},
8. {header:'Average Time', dataIndex:'averageTime', width:150, sortable: true}
9.],
10. bbar: pager, // <--- Barra de paginación
11. border: false,
12. stripeRows: true
13. });

```

The screenshot shows a window titled "Grid example". Inside, there is a data grid with the following data:

|   | City        | Visits | Page/Visits | Average Time |
|---|-------------|--------|-------------|--------------|
| 1 | Mexico city | 684    | 4.11        | 00:06:53     |
| 2 | La Victoria | 443    | 4.39        | 00:07:28     |
| 3 | Madrid      | 380    | 3.11        | 00:05:22     |
| 4 | Providencia | 204    | 3.83        | 00:08:20     |
| 5 | Bogota      | 204    | 3.26        | 00:04:57     |

Below the grid is a "Paging Toolbar" with the following controls and information:

- Left arrow, right arrow, first page, last page buttons.
- Page: 1 of 3
- Refresh button.
- Total: 1 - 5 of 14 Cities

Barra de paginación

El componente “Ext.PagingToolbar” se encarga de controlar la navegación entre las páginas, cuenta con botones para avanzar/retroceder página por página o avanzar hasta el final/inicio.

#### Enviar parámetros extra

Si en cada petición necesitamos mandar parámetros extra al servidor, podemos definirle al “store” la propiedad “baseParams” la cual recibe un objeto de la siguiente manera:

```

1. var store = new Ext.data.JsonStore({
2. url: 'paging.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: ['city','visits','pageVisits','averageTime'],
6. baseParams: {x:10,y:20} //<--- parámetros definidos
7. });

```

Al paginar la información automáticamente se mandarán los parámetros definidos en el código anterior, recordemos que estos parámetros normalmente nunca cambian.

Existen situaciones en las que necesitamos enviar un parámetro “z” en algún momento en específico o necesitamos modificar este parámetro con diferentes valores, por lo tanto se los mandamos cuando hacemos el “load” del store:

```

1. //parámetros variables
2. store.load({params:{z:10}});

```

Grid example

|   | City        | Visits | Page/Visits | Average Time |
|---|-------------|--------|-------------|--------------|
| 1 | Mexico city | 684    | 4.11        | 00:06:53     |
| 2 | La Victoria | 443    | 4.39        | 00:07:28     |
| 3 | Madrid      | 380    | 3.11        | 00:05:22     |
| 4 | Providencia | 204    | 3.83        | 00:08:20     |
| 5 | Bogota      | 204    | 3.26        | 00:04:57     |

## Error NaN en la barra de paginación

Al parecer esto nos resuelve el problema fácilmente, pero cuando actualizamos el explorador nos damos cuenta que aparecen algunos “NaN” en la barra de paginación en lugar de aparecer los números correctos, este error es muy común y sucede porque el componente no puede calcular la página en la que se encuentra, para resolverlo solamente necesitamos enviar los parámetros “start” y “limit” junto con el parámetro “z”.

1. //resolviendo el NaN
  2. store.load({params:{z:10,start:0,limit:5}});

Grid example

|   | City        | Visits | Page/Visits | Average Time |
|---|-------------|--------|-------------|--------------|
| 1 | Mexico city | 684    | 4.11        | 00:06:53     |
| 2 | La Victoria | 443    | 4.39        | 00:07:28     |
| 3 | Madrid      | 380    | 3.11        | 00:05:22     |
| 4 | Providencia | 204    | 3.83        | 00:08:20     |
| 5 | Bogota      | 204    | 3.26        | 00:04:57     |

## Error NaN solucionado

Hemos resuelto el problema del NaN, pero ahora cada vez que paginamos el parámetro "z" se está perdiendo, para solucionar esto podemos utilizar un evento ("beforechange") el cual se dispara antes de que el componente cambie de página, es decir antes de que se haga la llamada por medio de Ajax al servidor, este evento es el que necesitamos, porque aquí podemos agregar los parámetros que necesitemos, en este caso el parámetro "z".

```

1. pager.on('beforechange',function(bar,params){
2. params.z = 30;
3. });

```

Con esto es suficiente para que el parámetro "z" se envíe en cada cambio de página.

## Conclusiones

En este tema vimos como mostrar mucha información en partes, este componente es muy fácil de utilizar y sobre todo muy flexible, espero el funcionamiento haya quedado claro, si tienes dudas puedes inscribirte en el foro para poder platicar mejor, de igual forma puedes seguirnos en Twitter para estar actualizado en los nuevos tutoriales que vamos sacando.

## Formato a la información de las celdas

En este tema vamos a ver la forma en que podemos modificar el contenido de las celdas utilizando un "interceptor" el cual se ejecutará antes de que el contenido sea "renderizado", además veremos como cambiar el nombre de los parámetros que envía la barra de paginación al servidor.

Lo que vamos a realizar es una tabla que contenga información sobre algunas razas de perros, esta tabla tendrá una imagen, nombre, descripción y algunos otros campos más, si lo deseas puedes probar el ejemplo en ejecución.

|   | Picture                                                                             | Breed name                               | Description                                                                                                                                                                                                                                    | Agressive |
|---|-------------------------------------------------------------------------------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| 1 |   | <b>Beagle</b> - Ancient Greece           | The Beagle is a breed of small to medium-sized dog. A member of the Hound Group, it is similar in appearance to the Foxhound but smaller, with shorter legs and longer, softer ears.                                                           | No        |
| 2 |  | <b>German Shepherd</b> - Germany         | German Shepherds are a large-breed dog which generally are between 55 and 65 centimetres (22 and 26 in) at the withers and weigh between 22 and 40 kilograms (49 and 88 lb).                                                                   | Yes       |
| 3 |  | <b>Golden Retriever</b> - United Kingdom | The Golden Retriever is a breed of dog, historically developed as a gundog to retrieve shot waterfowl and upland game during hunting. As such they were bred to have a soft mouth to retrieve game undamaged and an instinctive love of water. | No        |
| 4 |  | Maltese - Central                        | The Maltese is a small breed of dog.                                                                                                                                                                                                           | No        |

Tabla con formato personalizado en las celdas

## Material de apoyo

Vamos a descargar el material de apoyo para que podamos seguir avanzando, después copia los archivos al servidor Web dentro de la carpeta "grids" en la que hemos estado trabajando.

Recuerden que estamos trabajando con Ext JS 3.0.0, así que si no la tienes puedes ir a descargarla y copiar los archivos necesarios dentro de la carpeta "curso" en el servidor Web.

## Información a mostrar

A continuación se muestra la información que el servidor nos regresará.

```

1. <?php
2. header("Content-Type: text/plain");
3.
4. $offset = isset($_POST['offset'])?$_POST['offset']:0;
5. $size = isset($_POST['size'])?$_POST['size']:5;

```

```

6.
7. $data = array(
8. array('breed'=>'Beagle','origin'=>'Ancient Greece','agressive'=>false,'image'=>'images/beagle.jpg','description'=>'The Beagle is a breed of small to medium-sized dog. A member of the Hound Group, it is similar in appearance to the Foxhound but smaller, with shorter legs and longer, softer ears.'),
9. array('breed'=>'German Shepherd','origin'=>'Germany','agressive'=>true,'image'=>'images/germanShep.jpg','description'=>'German Shepherds are a large-breed dog which generally are between 55 and 65 centimetres (22 and 26 in) at the withers and weigh between 22 and 40 kilograms (49 and 88 lb.).'),
10. array('breed'=>'Golden Retriever','origin'=>'United Kingdom','agressive'=>false,'image'=>'images/goldenRetriever.jpg','description'=>'The Golden Retriever is a breed of dog, historically developed as a gundog to retrieve shot waterfowl and upland game during hunting. As such they were bred to have a soft mouth to retrieve game undamaged and an instinctive love of water.'),
11. array('breed'=>'Maltese','origin'=>'Central Mediterranean Area','agressive'=>false,'image'=>'images/maltese.jpg','description'=>'The Maltese is a small breed of dog in the toy group, known for its silky white hair, though many owners of pet Maltese give them a short "puppy cut" for ease of grooming.'),
12. array('breed'=>'Rottweiler','origin'=>'Germany','agressive'=>false,'image'=>'images/rottweiler.jpg','description'=>'The Rottweiler, is a "medium to large size, stalwart dog" breed originating in Germany as a herding dog. It is a hardy and very intelligent breed.'),
13. array('breed'=>'St. Bernard','origin'=>'Italy / Switzerland','agressive'=>false,'image'=>'images/stbernard.jpg','description'=>'The St. Bernard Dog is a very large breed of dog, a working dog from the Swiss Alps, originally bred for rescue. The breed has become famous through tales of alpine rescues, as well as for its enormous size.'),
14. array('breed'=>'Whippet','origin'=>'England','agressive'=>false,'image'=>'images/whiteWhippet.jpg','description'=>'The Whippet is a breed of dog, of the sighthound family. They are active and playful and are physically similar to a small greyhound. Their popularity has led to the reuse of the Whippet name on a large number of things, from cars to cookies.'),
15. array('breed'=>'Chihuahua','origin'=>'Mexico','agressive'=>true,'image'=>'images/chihuahua.jpg','description'=>'Breed standards for this dog do not generally specify a height, only a weight and a description of their overall proportions. As a result, height varies more than within many other breeds.')
16.);
17.
18. $paging = array(
19. 'success'=>true,
20. 'total'=>count($data),
21. 'data'=> array_splice($data,$offset,$size)
22.);
23.
24. echo json_encode($paging);
25. ?>

```

En el código anterior la información está contenida en arreglos de PHP, para facilitar las cosas en este tutorial, normalmente esta información estaría en una base de datos o la podríamos obtener de un servicio Web (Web Service) o de cualquier otro lugar.

Lo que esta haciendo el código es recibir dos parámetros, estos serán usados para paginar, luego imprime en formato JSON únicamente los registros que le solicitamos.

### Definir el Namespace

El siguiente paso es definir el espacio de nombres que usaremos, no mencionaré el porqué de este paso porque ya lo hemos estudiado a detalle anteriormente.

```

1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.GridFormatTutorial = {
4. init: function(){
5. //aqui vamos a escribir el código inicial

```

```

6. }
7. }
8.
9. Ext.onReady(com.quizzpot.tutorial.GridFormatTutorial.init,com.quizzpot.tutorial.GridFormatTutorial
());

```

### **Creación del Store**

Ya vimos que el servidor nos regresa la información en formato JSON, por lo tanto vamos a utilizar el “JsonStore” por conveniencia.

```

1. var store = new Ext.data.JsonStore({
2. url: 'gridformat.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: ['breed','origin',{name:'agressive',type:'boolean'},'image','description']
6. });
7. store.load();

```

Ya debemos conocer el código anterior, pues no tiene nada de especial, simplemente define la URL donde se realizará la petición por medio de Ajax al servidor, los campos de los registros, el total que nos servirá para paginar la información y de donde sale la información de los registros (root).

### **Creación del paginador**

Vamos a paginar los resultados, para eso creamos el componente “PagingToolbar” de la siguiente manera:

```

1. var pager = new Ext.PagingToolbar({
2. store: store, // <--grid and PagingToolbar using same store
3. displayInfo: true,
4. displayMsg: '{0} - {1} of {2} Dog breeds',
5. emptyMsg: 'No dog breeds to display',
6. pageSize: 5
7. });

```

Ya hemos estudiado este componente en el tema anterior, por lo tanto debemos conocer el código anterior.

### **Crear el Grid y la Ventana contenedor**

Ahora creamos el Grid y la ventana donde se mostrará de la siguiente manera:

```

1. var grid = new Ext.grid.GridPanel({
2. store: store, //grid and PagingToolbar using same store
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'Picture', dataIndex:'image',width:150,sortable: true},
6. {header:'Breed name', dataIndex:'breed',width:140,sortable: true},
7. {header:'Description', dataIndex:'description',width:180},
8. {header:'Agressive', dataIndex:'agressive', width:60,sortable: true}
9.],
10. bbar: pager, // adding the pagingtoolbar to the grid
11. border: false,
12. stripeRows: true
13. });
14.
15. var win = new Ext.Window({
16. title: 'Grid example',
17. layout: 'fit',
18. width: 590,
19. height:450,
20. items: grid

```

```

21. });
22.
23. win.show();

```

En primer lugar se crea el Grid, con las columnas necesarias y estas columnas se relacionan con las propiedades del Store mediante la propiedad “indexData”, le asignamos la barra de paginación y luego creamos la ventana que contiene al grid, ya estudiamos a detalle el código anterior en temas pasados es por eso que no me voy a detener a explicar más, si tienes dudas puedes consultar los tutoriales anteriores.

|   | Picture                    | Breed name       | Description                           | Agressive |
|---|----------------------------|------------------|---------------------------------------|-----------|
| 1 | images/beagle.jpg          | Beagle           | The Beagle is a breed of small to me  | false     |
| 2 | images/germanShep.jpg      | German Shepherd  | German Shepherds are a large-breed    | true      |
| 3 | images/goldenRetriever.jpg | Golden Retriever | The Golden Retriever is a breed of c  | false     |
| 4 | images/maltese.jpg         | Maltese          | The Maltese is a small breed of dog   | false     |
| 5 | images/rottweiler.jpg      | Rottweiler       | The Rottweiler, is a "medium to large | false     |

Grid sin formato

Como se puede ver en la imagen ya aparece la información en la tabla, pero no aparece la imagen ni el formato que queremos además el paginate no esta funcionando correctamente, ya que siempre nos regresa los mismos registros.

### Cambiar los parámetros del paginate

Si regresamos a ver el código del servidor notamos que está recibiendo dos parámetros para paginate, uno de ellos es “offset” y “size”, si ya leiste el tema anterior sabrás que el componente “PagingToolbar” no envía esos parámetros, sino que envía “start” y “limit”, tenemos dos opciones, cambiar el código del servidor o de alguna manera cambiar el nombre de los parámetros a enviar por el paginate.

Si por razones externas a nuestra voluntad no podemos cambiar el servidor, entonces para cambiar el nombre de estos parámetros lo hacemos en el store usando la propiedad “paramNames” de la siguiente manera:

```

1. paramNames: {
2. start : 'offset', // The parameter name which specifies the start row
3. limit : 'size', // The parameter name which specifies number of rows to return
4. sort : 'sort', // The parameter name which specifies the column to sort on
5. dir : 'dir' // The parameter name which specifies the sort direction
6. }

```

Entonces el store quedaría de la siguiente manera:

```

1. var store = new Ext.data.JsonStore({
2. url: 'gridformat.php',
3. root: 'data',
4. totalProperty: 'total',
5. paramNames: {
6. start : 'offset', // The parameter name which specifies the start row.
7. limit : 'size', // The parameter name which specifies number of rows to return.
8. sort : 'sort', // The parameter name which specifies the column to sort on.
9. dir : 'dir' // The parameter name which specifies the sort direction.
10. },
11. fields: ['breed','origin',{name:'agressive',type:'boolean'},'image','description']
12. });
13. store.load();

```

Esta propiedad (paramNames) recibe un objeto donde sobrescribimos el nombre de los parámetros a enviar al servidor, nótese que también debemos definir el “sort” y “dir” que en futuros temas hablaremos al respecto.

Con esto es suficiente para que la paginación funcione adecuadamente.

| Grid example |                         |             |                                      |           |
|--------------|-------------------------|-------------|--------------------------------------|-----------|
|              | Picture                 | Breed name  | Description                          | Agressive |
| 1            | images/stbernard.jpg    | St. Bernard | The St. Bernard Dog is a very large  | false     |
| 2            | images/whiteWhippet.jpg | Whippet     | The Whippet is a breed of dog, of th | false     |
| 3            | images/chihuahua.jpg    | Chihuahua   | Breed standards for this dog do not  | true      |

Page 2 of 2 | 
6 - 8 of 8 Dog breeds

Cambio de nombre a los parámetros de la paginación

### Mostrar la imagen

Hasta ahora lo que tenemos en la columna de la imagen es la URL donde se encuentra la imagen a desplegar, lo que tenemos que hacer es utilizar la propiedad “renderer” del “ColumnModel” para modificar el contenido de la celda justo antes de que ésta sea renderizada.

```

1. var grid = new Ext.grid.GridPanel({
2. store: store,
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'Picture', dataIndex:'image',width:150,sortable: true, renderer: this.showImage}, // aq
 ui definimos el “interceptor”
6. {header:'Breed name', dataIndex:'breed',width:140,sortable: true },
7. {header:'Description', dataIndex:'description',width:180 },
8. {header:'Agressive', dataIndex:'agressive', width:60,sortable: true}
9.],
10. bbar: pager,

```

```

11. border: false,
12. stripeRows: true
13. });

```

Mediante la configuración “*renderer: this.showImage*” podemos definir una función, en este caso “*showImage*” que se ejecutará antes de que la celda sea “renderizada”, esta función debe regresar un String con el contenido que necesitamos desplegar en la celda, además también podemos generar el HTML.

```

1. com.quizzpot.tutorial.GridFormatTutorial = {
2. init: function(){
3. //... código removido para mayor visualización ...
4. },
5. //el interceptor
6. showImage: function(value, metaData, record, rowIndex, colIndex, store){
7. //retorna el contenido de la celda
8. return '';
9. }
10. }

```

|   | Picture                                                                             | Breed name       | Description                                | Agressive |
|---|-------------------------------------------------------------------------------------|------------------|--------------------------------------------|-----------|
| 1 |    | Beagle           | The Beagle is a breed of small to me false |           |
| 2 |   | German Shepherd  | German Shepherds are a large-breed true    |           |
| 3 |  | Golden Retriever | The Golden Retriever is a breed of c false |           |
| 4 |  | Maltese          | The Maltese is a small breed of dog false  |           |

Mostrar una imagen en la celda

La función que se configura en la propiedad “*renderer*” recibe seis parámetros, en este caso solo ocupamos solamente el “*value*” y el “*record*”, de acuerdo con la documentación los parámetros son los siguientes.

- “*value*” (Object) : Es la información original que se imprimirá en la celda.
- “*metadata*” (Object) : Un objeto en el cual se pueden configurar los siguientes atributos:
  - “*css*” (String) : El nombre de una clase CSS para ser agregada al elemento TD de la celda.
  - “*attr*” (String) : Un atributo HTML definido como String para ser aplicado al DIV contenedor en la celda (ejemplo: ‘style="color:red;"’).
- “*record*” (Ext.data.record) : El registro del Store de donde la información es extraída.
- “*rowIndex*” (Number) : El índice de la fila
- “*colIndex*” (Number) : El índice de la columna

- “store” (Ext.data.Store) : El objeto store de donde el registro es extraído.

### Dos campos en una misma celda

Lo que haremos ahora es poner dos campos del record dentro de una misma celda, además vamos a usar el parámetro “metaData” para ponerle un estilo y poder hacer que el texto pueda ocupar más de un solo renglón.

```

1. showBreed: function(value, metaData, record, rowIndex, colIndex, store){
2. metaData.attr = 'style="white-space:normal"';
3. return '<em class="name">' + value + ' - ' + record.get('origin');
4. }
```

Como puedes ver se imprime el valor original de la celda y le aplicamos una clase CSS que hace que el texto se ponga negrito, además estamos utilizando el “record” para acceder a otro campo, de esta manera podemos imprimir cualquier otro campo que necesitemos.

Por último tenemos que definir la propiedad “renderer” a la columna “name” de la siguiente manera:

```
1. {header:'Breed name', dataIndex:'breed', width:140, sortable: true, renderer: this.showBreed},
```

|   | Picture                                                                             | Breed name                               | Description                          | Agressive |
|---|-------------------------------------------------------------------------------------|------------------------------------------|--------------------------------------|-----------|
| 1 |    | <b>Beagle</b> - Ancient Greece           | The Beagle is a breed of small to me | false     |
| 2 |   | <b>German Shepherd</b> - Germany         | German Shepherds are a large-breed   | true      |
| 3 |  | <b>Golden Retriever</b> - United Kingdom | The Golden Retriever is a breed of c | false     |
| 4 |  | Maltese - Central                        | The Maltese is a small breed of dog  | false     |

Varios campos en una misma celda

### Corregir la descripción

Si has notado, la descripción es mas larga que la celda y no se ve completa, lo que tenemos que hacer para que el texto se despliegue en renglones es modificar el CSS por medio del parámetro “metadata”, igual como lo hicimos en el paso anterior con el campo “breed”.

```

1. showDescription: function(value, metaData){
2. metaData.attr = 'style="white-space:normal"';
3. return value;
4. }
```

Con esto es suficiente para que la columna “Description” se pueda ver correctamente, recuerda asignarle esta función en la definición de la columna utilizando la propiedad “renderer”.

|   | Picture                                                                           | Breed name                               | Description                                                                                                                                                                                                                                    | Aggressive |
|---|-----------------------------------------------------------------------------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| 1 |  | <b>Beagle</b> - Ancient Greece           | The Beagle is a breed of small to medium-sized dog. A member of the Hound Group, it is similar in appearance to the Foxhound but smaller, with shorter legs and longer, softer ears.                                                           | false      |
| 2 |  | <b>German Shepherd</b> - Germany         | German Shepherds are a large-breed dog which generally are between 55 and 65 centimetres (22 and 26 in) at the withers and weigh between 22 and 40 kilograms (49 and 88 lb).                                                                   | true       |
| 3 |  | <b>Golden Retriever</b> - United Kingdom | The Golden Retriever is a breed of dog, historically developed as a gundog to retrieve shot waterfowl and upland game during hunting. As such they were bred to have a soft mouth to retrieve game undamaged and an instinctive love of water. | false      |
| 4 |  | <b>Maltese</b> - Central                 | The Maltese is a small breed of dog                                                                                                                                                                                                            | false      |

Page 1 of 2

1 - 5 of 8 Dog breeds

Asignando estilos a la celda

### Cambiando el texto booleano

Por último vamos a cambiar el “true” y “false” de la columna “aggressive” por algo más amigable.

1. showAggressive: function(value,metaData){
2. metaData.attr = value?'style="color:#f00"':'style="color:#0a0"';
3. return value?'Yes':'No';
4. }

El código anterior regresa “Yes” o “No” además en caso de ser “Yes” le asigna color rojo al texto y verde para cuando es un “No”.

|   | Picture                                                                             | Breed name                               | Description                                                                                                                                                                                                                                    | Aggressive |
|---|-------------------------------------------------------------------------------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| 1 |  | <b>Beagle</b> - Ancient Greece           | The Beagle is a breed of small to medium-sized dog. A member of the Hound Group, it is similar in appearance to the Foxhound but smaller, with shorter legs and longer, softer ears.                                                           | No         |
| 2 |  | <b>German Shepherd</b> - Germany         | German Shepherds are a large-breed dog which generally are between 55 and 65 centimetres (22 and 26 in) at the withers and weigh between 22 and 40 kilograms (49 and 88 lb).                                                                   | Yes        |
| 3 |  | <b>Golden Retriever</b> - United Kingdom | The Golden Retriever is a breed of dog, historically developed as a gundog to retrieve shot waterfowl and upland game during hunting. As such they were bred to have a soft mouth to retrieve game undamaged and an instinctive love of water. | No         |
| 4 |  | <b>Maltese</b> - Central                 | The Maltese is a small breed of dog                                                                                                                                                                                                            | No         |

Page 1 of 2

1 - 5 of 8 Dog breeds

Modificando los estilos y textos a depilar en la celda

## Conclusiones

Hemos visto como el Framework nos permite controlar completamente el contenido de una celda, podemos utilizar este método para realizar lo que sea necesario en nuestras aplicaciones.

## Utilizando fechas y tiempo en un Grid

Desplegar una fecha o una hora en un grid es algo muy común, si no sabemos como utilizar correctamente este tipo de dato podemos meternos en problemas y puede darnos muchos dolores de cabeza.

En este tema explicaré como podemos mostrar las fechas en diferentes formatos, además veremos como podemos ordenar correctamente ascendente o descendente las fechas contenidas en una tabla.

El ejercicio que haremos es una simple “grilla” donde se mostrarán algunas fechas aleatorias en diferentes formatos, puedes ver el ejemplo del tutorial y probar su funcionalidad, recuerda que usaremos Ext JS 3.0.0.

|   | Name     | From     | To       | Time        | From ISO date | From Unix Time |
|---|----------|----------|----------|-------------|---------------|----------------|
| 1 | John Doe | Dec/1982 | Mar/2001 | 03:52:04 pm | Sat 06 Jun    | Jan '05        |
| 2 | Crystel  | Sep/1986 | Oct/1996 | 11:26:42 am | Mon 12 Nov    | Apr '78        |
| 3 | Sasha    | Nov/1970 | Nov/1987 | 11:12:33 am | Tue 24 Sep    | Oct '00        |
| 4 | Peter    | Aug/1977 | Feb/2007 | 03:56:17 pm | Mon 16 Aug    | Nov '72        |
| 5 | Carl     | May/2005 | Jan/1997 | 02:52:10 am | Thu 10 Jan    | Apr '82        |
| 6 | Ronaldo  | Aug/2000 | Nov/2007 | 11:42:03 pm | Wed 17 Feb    | Mar '04        |
| 7 | Jenny    | Nov/1981 | Feb/1979 | 06:44:53 pm | Thu 04 Sep    | Sep '90        |
| 8 | Gina     | Oct/2003 | Oct/1979 | 03:15:18 am | Fri 22 Sep    | Mar '78        |
| 9 | Eddy     | Jan/2007 | Apr/1982 | 03:58:14 am | Sat 21 Apr    | Dec '81        |

Ejercicio final

## Material de apoyo

Antes de continuar adelante es necesario descargar el material de apoyo y copiar los archivos al servidor Web que instalamos al inicio de este curso.

## Información a mostrar

La información que desplegaremos en el Grid se muestra a continuación:

```
1. <?php
2. header("Content-Type: text/plain");
3.
4. $data = array(
5. array('name'=>'John Doe','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
6. array('name'=>'Crystel','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
7. array('name'=>'Sasha','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
8. array('name'=>'Peter','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
9. array('name'=>'Carl','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
```

```

10. array('name'=>'Ronaldo','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-
 Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
11. array('name'=>'Jenny','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-
 Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
12. array('name'=>'Gina','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-
 Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U')),
13. array('name'=>'Eddy','from'=>randomDate('m-d-Y'), 'to'=> randomDate('j-M-
 Y'), 'time'=>randomDate('h:i:s a'), 'iso'=>randomDate('c'), 'unix'=>randomDate('U'))
14.);
15.
16. $response = array(
17. 'success'=>true,
18. 'total'=>count($data),
19. 'data'=> $data
20.);
21.
22. echo json_encode($response);
23.
24. function randomDate($format){
25. return date($format,rand(0,time()));
26. }
27. ?>

```

Como se pueden dar cuenta se están generando las fechas de manera aleatoria, así que no esperen ver las mismas fechas siempre, lo único que es constante es el nombre de las personas.

También es importante notar que las fechas son generadas en diferentes formatos, puedes ver que también usamos el formato de “unix”, especificando los segundos transcurridos a partir del 1ro de enero de 1970.

### Namespace

Una vez definida la información vamos a comenzar a trabajar del lado de cliente, por lo tanto necesitamos definir el espacio de nombre que usaremos para este tutorial.

```

1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.DatesTutorial = {
4. init: function(){
5. //el código va aqui
6. }
7. }
8.
9. Ext.onReady(com.quizzpot.tutorial.DatesTutorial.init,com.quizzpot.tutorial.DatesTutorial);

```

Ya sabemos el por qué de utilizar un namespace robusto, si no lo recuerdas te sugiero repasar el tema correspondiente.

### Crear el Store y el Grid

Vamos a crear el Store capaz de procesar la información regresada por el servidor, la cual esta en formato JSON.

```

1. var store = new Ext.data.JsonStore({
2. url: 'dates.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: [
6. 'name',
7. 'from',
8. 'to',
9. 'time',

```

```

10. 'iso',
11. 'unix'
12.
13.];
14. });
15. store.load();

```

El código anterior ya lo conocemos muy bien, lo hemos estudiado reiteradamente en los temas anteriores, no tiene nada especial, ahora vamos a crear la tabla e introducirla dentro de una ventana de la siguiente manera:

```

1. var grid = new Ext.grid.GridPanel({
2. store: store,
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'Name', dataIndex:'name', sortable: true},
6. {header:'From', dataIndex:'from', sortable: true},
7. {header:'To', dataIndex:'to', sortable: true},
8. {header:'Time', dataIndex:'time', sortable: true},
9. {header:'From ISO date', dataIndex:'iso', sortable: true},
10. {header:'From Unix Time', dataIndex:'unix', sortable: true}
11.],
12. viewConfig: {
13. forceFit: true
14. },
15. border: false,
16. stripeRows: true
17. });
18.
19. var win = new Ext.Window({
20. title: 'Grid example',
21. layout: 'fit',
22. width: 510,
23. height:350,
24. items: grid
25. });
26.
27. win.show();

```

Lo único que es importante resaltar en el código anterior es que estamos usando la propiedad “viewConfig”, esta propiedad nos permite configurar el “view” del grid que estamos creando, para este caso solamente estamos forzando a las columnas que se distribuyan sobre el grid para que todas sean mostradas.

|   | Name     | From | To          | Time        | From ISO date | From Unix Time |
|---|----------|------|-------------|-------------|---------------|----------------|
| 1 | John Doe |      | 17-Oct-1990 | 05:39:38 pm | 2003-07-31T2  | 586227096      |
| 2 | Crystel  |      | 6-Jul-2002  | 02:15:20 am | 2006-08-15T2  | 112440154      |
| 3 | Sasha    |      | 8-Jun-1988  | 07:56:43 am | 1996-12-15T1  | 922138726      |
| 4 | Peter    |      | 13-May-1972 | 05:02:53 pm | 1992-12-24T0  | 1107838229     |
| 5 | Carl     |      | 20-Dec-1982 | 12:59:08 pm | 1970-05-23T0  | 348476902      |
| 6 | Ronaldo  |      | 4-Nov-1989  | 11:13:20 pm | 1988-07-05T0  | 270077214      |
| 7 | Jenny    |      | 24-Jun-1996 | 08:39:40 am | 1989-03-18T0  | 244527826      |
| 8 | Gina     |      | 22-Oct-1978 | 02:53:35 pm | 1985-01-08T1  | 635536273      |
| 9 | Eddy     |      | 29-Sep-1970 | 07:02:01 am | 1989-01-05T1  | 1170778824     |

Un grid con configuraciones comunes

### Problemas con el manejo de las fechas y tiempos

Si lo han notado, por alguna extraña razón no está apareciendo la información de una de las columnas, además si ordenamos ascendente o descendenteamente alguna de las columnas que muestran una fecha o tiempo, no se ordenan correctamente.

|   | Name     | From | To          | Time        | From ISO date | From Unix Time |
|---|----------|------|-------------|-------------|---------------|----------------|
| 1 | Sasha    |      | 8-Jun-1988  | 07:56:43 am | 1996-12-15T1  | 922138726      |
| 2 | Crystel  |      | 6-Jul-2002  | 02:15:20 am | 2006-08-15T2  | 112440154      |
| 3 | Ronaldo  |      | 4-Nov-1989  | 11:13:20 pm | 1988-07-05T0  | 270077214      |
| 4 | Eddy     |      | 29-Sep-1970 | 07:02:01 am | 1989-01-05T1  | 1170778824     |
| 5 | Jenny    |      | 24-Jun-1996 | 08:39:40 am | 1989-03-18T0  | 244527826      |
| 6 | Gina     |      | 22-Oct-1978 | 02:53:35 pm | 1985-01-08T1  | 635536273      |
| 7 | Carl     |      | 20-Dec-1982 | 12:59:08 pm | 1970-05-23T0  | 348476902      |
| 8 | John Doe |      | 17-Oct-1990 | 05:39:38 pm | 2003-07-31T2  | 586227096      |
| 9 | Peter    |      | 13-May-1972 | 05:02:53 pm | 1992-12-24T0  | 1107838229     |

Errores al ordenar las columnas de fecha y tiempo

Como se aprecia en la imagen anterior, al ordenar la columna "To" de manera ascendente vemos que los meses de "Noviembre" y "Diciembre" se encuentran antes del mes de "Enero" o "Mayo", esto es totalmente incorrecto puesto que en lugar de ordenarlos tomando en cuenta que son fechas, simplemente los está ordenando por orden alfabético como si se tratases de palabras comunes, lo mismo sucede con la columna "Time" porque el tiempo está en formato de 12 horas así que hace el ordenamiento incorrectamente.

|   | Name     | From | To          | Time ▾      | From ISO date | From Unix Time |
|---|----------|------|-------------|-------------|---------------|----------------|
| 1 | Carl     |      | 20-Dec-1982 | 12:59:08 pm | 1970-05-23T0  | 348476902      |
| 2 | Ronaldo  |      | 4-Nov-1989  | 11:13:20 pm | 1988-07-05T0  | 270077214      |
| 3 | Jenny    |      | 24-Jun-1996 | 08:39:40 am | 1989-03-18T0  | 244527826      |
| 4 | Sasha    |      | 8-Jun-1988  | 07:56:43 am | 1996-12-15T1  | 922138726      |
| 5 | Eddy     |      | 29-Sep-1970 | 07:02:01 am | 1989-01-05T1  | 1170778824     |
| 6 | John Doe |      | 17-Oct-1990 | 05:39:38 pm | 2003-07-31T2  | 586227096      |
| 7 | Peter    |      | 13-May-1972 | 05:02:53 pm | 1992-12-24T0  | 1107838229     |
| 8 | Gina     |      | 22-Oct-1978 | 02:53:35 pm | 1985-01-08T1  | 635536273      |
| 9 | Crystel  |      | 6-Jul-2002  | 02:15:20 am | 2006-08-15T2  | 112440154      |

No ordena correctamente la hora

Otro problema es que estamos mostrando una fecha en milisegundos, un usuario no podrá interpretar dicha información (Ni siquiera nosotros los desarrolladores).

### Solución al problema de ordenación

Para solucionar los problemas anteriores y dar un formato más entendible para el usuario necesitamos definir en el “Store” que usaremos, los campos de tipo “date”.

```

1. var store = new Ext.data.JsonStore({
2. url: 'dates.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: [
6. 'name',
7. {name:'from', type:'date'},
8. {name:'to', type:'date'},
9. {name:'time', type:'date'},
10. {name:'iso', type:'date'},
11. {name:'unix', type:'date'}
12.]
13. });
14. store.load();

```

Si actualizamos la página veremos que ocurre algo inesperado:

|   | Name     | From | To | Time | From ISO date | From Unix Time |
|---|----------|------|----|------|---------------|----------------|
| 1 | John Doe |      |    |      |               |                |
| 2 | Crysfel  |      |    |      |               |                |
| 3 | Sasha    |      |    |      |               |                |
| 4 | Peter    |      |    |      |               |                |
| 5 | Carl     |      |    |      |               |                |
| 6 | Ronaldo  |      |    |      |               |                |
| 7 | Jenny    |      |    |      |               |                |
| 8 | Gina     |      |    |      |               |                |
| 9 | Eddy     |      |    |      |               |                |

Se pierde la información de las columnas

Ha desaparecido la información que estábamos desplegando anteriormente, la pregunta es ¿por qué? ¿qué hemos hecho mal? Solamente le hemos especificado a cada campo en el “store” el tipo de dato “date”.

La respuesta a las incógnitas anteriores es sencilla y ocurre porque no hemos definido el formato de la fecha o tiempo que viene del servidor, por lo tanto al convertir de “String” a “Date” sucede un error. Para solucionar esto necesitamos definir el formato por medio de la propiedad “dateFormat” de la siguiente manera:

```

1. var store = new Ext.data.JsonStore({
2. url: 'dates.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: [
6. 'name',
7. {name:'from', type:'date', dateFormat:'m-d-Y'},
8. {name:'to', type:'date', dateFormat:'j-M-Y'},
9. {name:'time', type:'date', dateFormat:'h:i:s a'},
10. {name:'iso', type:'date', dateFormat:'c'},
11. {name:'unix', type:'date', dateFormat:'U'}
12.]
13. });
14. store.load();

```

Si actualizamos el navegador donde estamos trabajando veremos algo semejante a la siguiente imagen.

|   | Name     | From           | To             | Time          | From ISO date   | From Unix Time |
|---|----------|----------------|----------------|---------------|-----------------|----------------|
| 1 | John Doe | Sun Feb 14 15  | Thu May 19 15  | Wed Jul 15 20 | Thu Jul 23 198  | Thu Jul 16 198 |
| 2 | Crystel  | Thu Nov 25 15  | Thu Dec 27 15  | Wed Jul 15 20 | Fri Nov 11 197  | Tue Mar 23 19  |
| 3 | Sasha    | Sat Jun 04 199 | Sat Nov 02 19  | Wed Jul 15 20 | Wed Feb 12 11   | Sun Dec 20 199 |
| 4 | Peter    | Mon Dec 29 21  | Tue Oct 28 19  | Wed Jul 15 20 | Thu May 03 15   | Thu Jul 27 198 |
| 5 | Carl     | Fri Nov 17 200 | Sun Dec 18 15  | Wed Jul 15 20 | Fri Jul 07 2000 | Mon Feb 17 19  |
| 6 | Ronaldo  | Mon Sep 06 11  | Wed Feb 13 11  | Wed Jul 15 20 | Tue Dec 08 15   | Thu Jul 23 199 |
| 7 | Jenny    | Mon Mar 06 15  | Mon Mar 13 20  | Wed Jul 15 20 | Thu Aug 07 21   | Sat Nov 20 199 |
| 8 | Gina     | Wed Jun 18 15  | Sat Apr 09 199 | Wed Jul 15 20 | Tue Sep 17 20   | Thu Feb 26 199 |
| 9 | Eddy     | Mon Mar 20 20  | Tue Jun 08 20  | Wed Jul 15 20 | Sun Nov 08 15   | Sat Jan 04 199 |

Fechas en formato ISO

Nota que ahora todas las columnas presentan la información en un mismo formato, el formato por defecto es el “iso”, inclusive para los campos en los que solamente teníamos el tiempo.

Si ordenamos nuevamente las columnas podremos ver como ahora si lo hace correctamente tanto los meses como los días, horas, años, etc.

### Cambiar el formato de las celdas

Ya sabemos que por medio de la propiedad “renderer” de las columnas podemos cambiar el contenido de las celdas, así que vamos a modificar el formato con el que se está mostrando las fechas utilizando una utilería del Framework.

```

1. var grid = new Ext.grid.GridPanel({
2. store: store,
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'Name', dataIndex:'name', sortable: true},
6. {header:'From', dataIndex:'from', sortable: true, renderer: Ext.util.Format.dateRenderer('M/Y')}
7. ,
8. {header:'To', dataIndex:'to', sortable: true, renderer: Ext.util.Format.dateRenderer('M/Y')},
9. {header:'Time', dataIndex:'time', sortable: true, renderer: Ext.util.Format.dateRenderer('h:i:s a')}
10. },
11. {header:'From ISO date', dataIndex:'iso', sortable: true, renderer: Ext.util.Format.dateRender
12. er('D d M')},
13. {header:'From Unix Time', dataIndex:'unix', sortable: true, renderer: Ext.util.Format.dateRend
14. er('M \\'y')}
15.],
16. viewConfig: {
17. forceFit: true
18. },
19. border: false,
20. stripeRows: true
21. });

```

Utilizando el método “dateRenderer” del objeto “Ext.util.Format” podemos cambiar el formato de una manera muy sencilla, así no creamos una función para cada columna, simplemente definimos el formato que necesitamos.

|   | Name     | From     | To       | Time        | From ISO date | From Unix Time |
|---|----------|----------|----------|-------------|---------------|----------------|
| 1 | John Doe | Jul/2004 | Aug/1984 | 01:41:35 am | Sat 13 Feb    | May '82        |
| 2 | Crysfel  | Aug/1994 | Dec/1988 | 10:27:54 am | Sun 16 Oct    | Jan '78        |
| 3 | Sasha    | Sep/1995 | Oct/1976 | 05:33:09 pm | Tue 03 Sep    | May '98        |
| 4 | Peter    | Jul/1984 | Aug/1976 | 11:19:47 pm | Wed 24 Oct    | Nov '79        |
| 5 | Carl     | May/2007 | Feb/1993 | 10:24:55 pm | Mon 18 May    | Apr '91        |
| 6 | Ronaldo  | Nov/1995 | May/1979 | 05:04:20 am | Sun 14 Jul    | Jul '95        |
| 7 | Jenny    | Jan/1983 | Nov/1977 | 01:46:52 pm | Tue 05 Sep    | Mar '91        |
| 8 | Gina     | Mar/1990 | Jun/1996 | 09:12:24 am | Tue 13 Dec    | Oct '03        |
| 9 | Eddy     | Sep/1988 | Sep/2002 | 01:02:18 am | Wed 24 Aug    | Aug '72        |

Formato correcto en el grid

Ahora si podemos mostrar la información de una manera muy atractiva, además puede ser ordenada correctamente.

### Conclusiones

Es muy común que cuando trabajamos con fechas sucedan los errores que vimos, he recibido muchos comentarios solicitando ayuda con respecto a este tema, es por eso que creo que es importante despejar todas estas dudas.

## Editar la fila de un Grid en un formulario

En este tutorial voy a explicar como podemos editar una fila de un Grid utilizando un formulario en una ventana por separado, vamos a llenar los campos del formulario con la información que tenemos capturada en el Grid.

### Demostración

Vamos a desplegar en un Grid las películas que han sido más vistas, al dar doble clic sobre una de las filas vamos abrir una ventana que contenga un formulario con campos que serán llenados con la información que tengamos en el Grid, puedes probar la demostración la cual es semejante a la siguiente imagen:

The screenshot shows a 'Edit Movie' dialog box overlaid on a movie grid. The dialog contains the following fields:

- Title: G-Force
- Year: 2009
- Weekend: 32.2
- All Time: 32.2
- Image: images/gforce.jpg

The movie grid in the background lists several movies, including 'THE WORLD NEEDS BIGGER HEROES' (G-Force) at the top.

Ejemplo Final

## Material de apoyo

Para continuar es necesario descargar el material de apoyo donde ya esta creada el Grid que muestra la información. Recuerda que tienes que copiar los archivos descargados al servidor Web que instalamos al inicio de este curso.

Si en este momento ejecutas en tu explorador el material de apoyo, podrás ver la información en el Grid.

| USA Weekend Box-Office Summary |                                                                                   |                                                                                                                                                                                                                                                                |      |         |          |  |
|--------------------------------|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|---------|----------|--|
|                                | Image                                                                             | Title                                                                                                                                                                                                                                                          | Year | Weekend | All Time |  |
| 1                              |  | <b>G-Force</b><br>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam sapien nibh, dictum pellentesque cursus eu, fringilla at elit. Integer volutpat, lorem consequat auctor pellentesque, arcu urna mattis lectus, sed tempus massa leo sed ante. | 2009 | \$32.2M | \$32.2M  |  |
| 2                              |  | <b>Harry Potter and the Half-Blood Prince</b><br>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam sapien nibh, dictum pellentesque                                                                                                               | 2009 | \$30M   | \$222M   |  |

Material de apoyo

También se pueden paginar las filas de cinco en cinco. Si tienes alguna duda en cuanto al código necesario para realizar lo anterior mencionado, te recomiendo leer los temas anteriores donde se explica a detalle como realizarlo.

### Doble clic en una fila

Necesitamos lograr que cuando el usuario de doble clic sobre una fila se abra una ventana con un formulario para poder editar la información, para esto necesitamos crear un "listener" del evento "rowdblclick" del grid. Al final del método "init" vamos a crear este "listener" de la siguiente manera:

1. grid.on('rowdblclick',this.editMovie);

Con la instrucción anterior logramos hacer que al dar doble clic sobre una fila (row) se ejecuta la función "editMovie", así que es necesario crear esta función:

```
1. Ext.ns('com.quizzpot.tutorial');
2.
3. com.quizzpot.tutorial.GridFormTutorial = {
4. init: function(){
5. //... código removido
6.
7. //TODO: create a listener for the "rowdblclick" event here
8. grid.on('rowdblclick',this.editMovie); //<-- creamos el listener
9. },
10.
11. //éste método se ejecuta cuando se da doble clic sobre una fila
12. editMovie: function(grid,index,event){
13. //aqui vamos a escribir el código donde creamos
14. //la ventana y llenamos el formulario.
15. },
16.
17. image: function(value,metadata,record){
18. //... código removido
19. },

```

```

20.
21. title: function(value, metadata, record){
22. //... código removido
23. },
24.
25. money: function(value){
26. //código removido
27. }
28.
29.
30. Ext.onReady(com.quizzpot.tutorial.GridFormTutorial.init,com.quizzpot.tutorial.GridFormTutorial);

```

Es importante mencionar los parámetros que está recibiendo la función “editMovie”, el primero es el grid sobre el cual se dio doble clic, el segundo es el índice en el store donde se encuentra la información que despliega la fila y el tercer parámetro es el evento ocurrido.

### Obtener el record del store

Con los parámetros recibidos es suficiente para obtener el “record” con la información a editar, utilizando el “grid” obtenemos el “store” y utilizando el “index” podemos sacar el “record” que necesitamos del “store”.

```

1. //este código va en la función “editMovie”
2. var record = grid.getStore().getAt(index);
3.
4. Ext.Msg.alert('Test','Title: '+record.get('title'));//alert temporal

```

Al actualizar el navegador y dar doble clic sobre alguna fila del grid podremos ver algo semejante a la siguiente imagen:



Obtener el record del store

### Crear el formulario de edición

Ahora vamos a crear el formulario que utilizaremos para editar la información que tenemos en el “record” y que fue sacada del grid.

```

1. var form = new Ext.form.FormPanel({
2. width:270,
3. bodyStyle:'margin-left:10px;',
4. border:false,
5. labelWidth: 80, //Con esto hacemos mas pequeñas las etiquetas
6. defaults: {
7. xtype:'textfield',
8. width:150
9. },
10. items:[
11. {fieldLabel:'Title',id:'title'},
12. {xtype:'combo', fieldLabel:'Year', id:'year' ,triggerAction:'all', store:[2009,2008,2007,2006]},
13. {xtype:'numberfield',fieldLabel:'Weekend',id:'weekend'},
14. {xtype:'numberfield',fieldLabel:'All Time',id:'allTime'},
15. {fieldLabel:'Image',id:'img'}

```

```
16.]
17.});
```

Ya hemos estudiado como crear formularios, si tienes alguna duda al respecto puedes ir a repasar el tema correspondiente. Este formulario no tiene nada especial, solamente se están creando cinco campos, dos de ellos son cajas de texto que aceptan cualquier caracter, los otros dos son cajas de texto que aceptan solamente números y el último campo es un combo para poder seleccionar el año.

Del código anterior es importante resaltar que el "id" de los campos se llamen igual que los campos que contiene el "record" que se definió en el store, esto es importante para que podamos llenarlos de una manera muy sencilla, más adelante veremos como hacerlo, por ahora solamente asegúrate que tengan el mismo nombre.

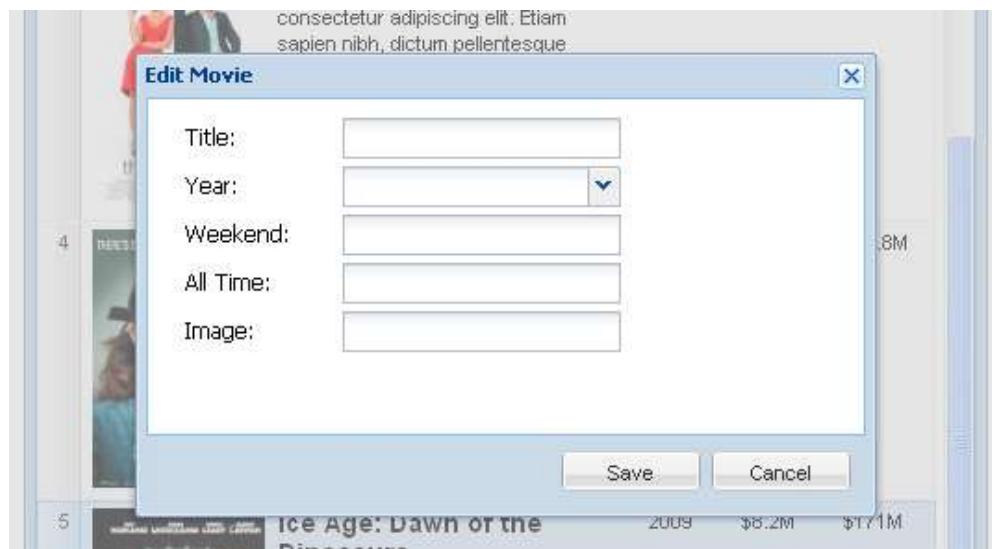
### Crear la ventana contenedora del formulario

Ya creamos el formulario, pero aún no se muestra en pantalla, es por eso que vamos a crear una ventana que contenga el formulario anterior.

```
1. var win = new Ext.Window({
2. title: 'Edit Movie',
3. width:400,
4. height:250,
5. modal: true,
6. bodyStyle: 'padding:10px;background-color:#fff',
7. buttons: [{text:'Save'}, {text:'Cancel'}],
8. items: [form]
9. });
10. win.show();
```

El código anterior debe ser familiar para ti, ya que en repetidas ocasiones lo hemos hecho, pero si existe alguna duda es mejor aclararla en el tutorial donde hablamos acerca de las ventanas.

Si actualizamos el explorador y damos doble clic sobre alguna fila veremos algo como la siguiente imagen.



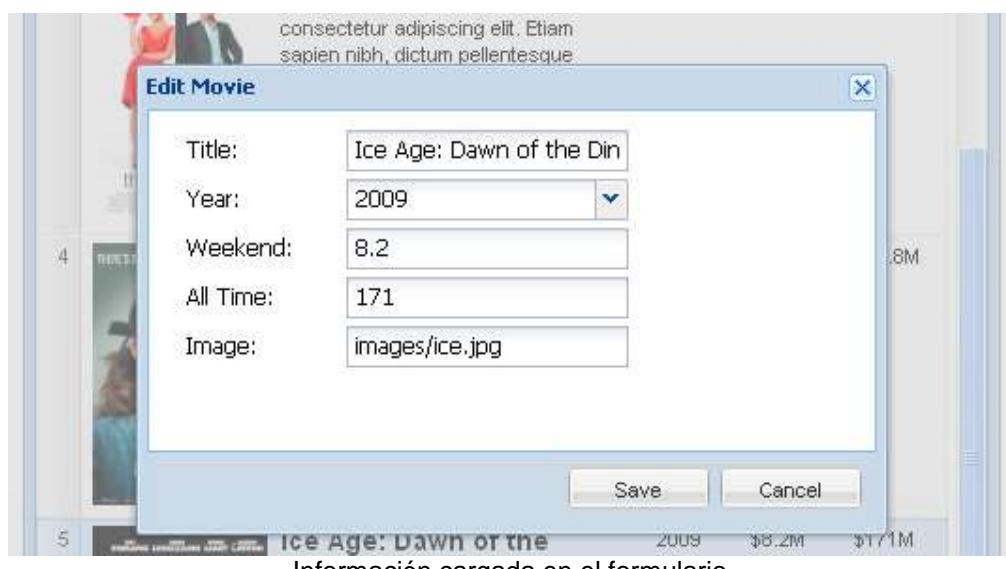
Formulario en una ventana

### Llenar los campos del formulario

Hasta ahora aparece el formulario vacío, en este paso vamos a llenarlo de una manera muy sencilla.

```
1. form.getForm().loadRecord(record);
```

Con la línea anterior es suficiente para que la información de la fila en la que hicimos doble clic se cargue en los campos del formulario, esto es posible ya que nombramos el “id” de cada campo igual a los campos del registro.



Información cargada en el formulario

### Vista previa de la imagen

Para darle un toque especial a nuestro formulario vamos a desplegar en la parte izquierda la imagen de cada película a editar, para esto vamos a crear un panel que contenga una imagen la cual podremos modificarle su “src” mediante su “id”.

```
1. var preview = new Ext.Panel({
2. width:91,
3. height:140,
4. html: '' //imagen vacía
5. });
6.
7. var win = new Ext.Window({
8. title: 'Edit Movie',
9. width:400,
10. height:250,
11. modal: true,
12. bodyStyle: 'padding:10px;background-color:#fff',
13. buttons: [{text:'Save'}, {text:'Cancel'}],
14. items: [preview,form]
15.});
16. win.show();
```

Si actualizas el explorador verás que el formulario no se ve bien, esto es por que los paneles se van posicionando como en una pila.



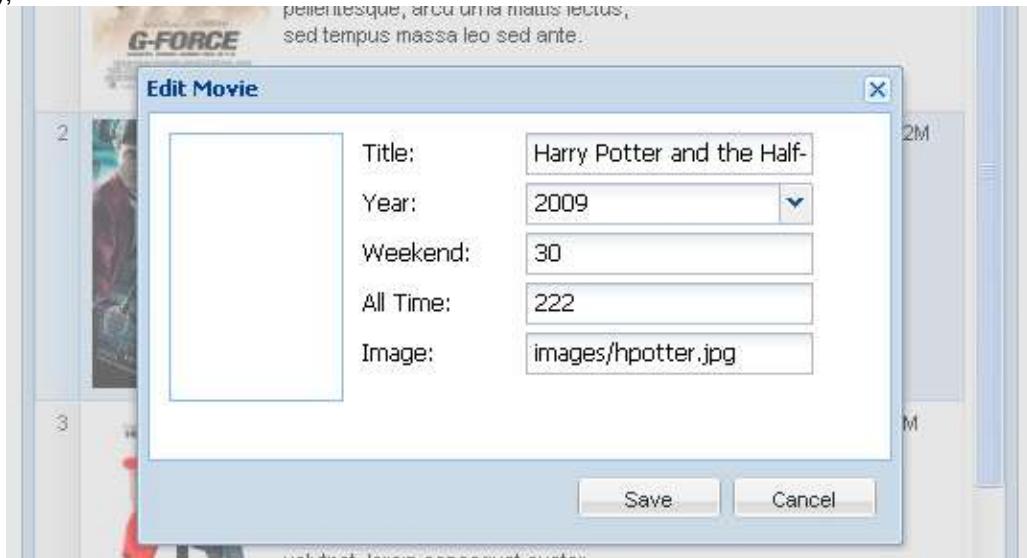
Se desacomoda el formulario

Para solucionar esto es necesario cambiar el “layout” de la ventana, en este caso vamos a utilizar el “column” para crear columnas, no quiero profundizar en el tema de los “layout” porque en el futuro hablaré en detalle sobre esto, por ahora simplemente ten en mente que al cambiar el layout a columnas solucionaremos el problema.

```

1. var win = new Ext.Window({
2. layout: 'column', //cambiamos la manera en que se posicionan los paneles
3. title: 'Edit Movie',
4. width:400,
5. height:250,
6. modal: true,
7. bodyStyle: 'padding:10px;background-color:#fff',
8. buttons: [{text:'Save'}, {text:'Cancel'}],
9. items: [preview,form]
10. });

```



Utilizando columnas para mostrar correctamente el formulario

Lo siguiente es modificar el “src” dinámicamente para que aparezca la imagen dentro del contenedor.

```
1. Ext.get('preview').dom.src = record.get('img');
```

Recuerda colocar la línea anterior después de mostrar la ventana que “renderiza” el formulario y la ventana.



Mostrar la imagen inicial

Por último necesitamos refrescar la imagen al actualizar el campo “img” del formulario, para esto necesitamos poner un “listener” al evento “blur” del campo “img”.

1. Ext.getCmp('img').on('blur',function(field){
2.     Ext.get('preview').dom.src = field.getValue();
3. });

#### Guardar la información del formulario

Para guardar la información de un formulario simplemente ejecutamos el método “submit”, con esto hacemos que el formulario envíe la información contenida en sus campos al servidor mediante Ajax, en esta ocasión no voy a mostrar como hacerlo pues quiero dedicar un tema exclusivamente a esto.

#### Conclusiones

El día de hoy vimos como llenar un formulario con información de un “record”, también aprendimos como crear columnas en una ventana, si tienes alguna duda puedes preguntar en el foro donde la comunidad puede ayudarte, además recuerda inscribirte a las Feeds mediante tu lector preferido de RSS o mediante correo electrónico.

## Grid con columnas dinámicas

En esta ocasión veremos cómo podemos hacer nuestros “Grids” un poco más dinámicos, esto lo haremos al cargar todo la información desde nuestra base de datos, incluyendo la información necesaria para los “headers” del “Grid”.

En este tutorial veremos cómo podemos cargar dinámicamente la propiedad “header” de un “Grid”, con la finalidad de hacer nuestro código de una manera más eficiente y dinámico dando posibilidad de poder definir las columnas de manera dinámica.

Esta es una muestra de lo que se obtendrá al final de este tutorial. Recuerda que puedes descargar el código fuente si es necesario.

| Name   | Lastname | Age | Sex    | ZipCode |
|--------|----------|-----|--------|---------|
| John   | Smith    | 20  | Male   | 50300   |
| Nicole | Summer   | 22  | Female | 55687   |

Resultado Final

### La base de datos

En esta ocasión la información que usaremos está en una tabla llamada “grid” la cual contiene la información necesaria para cumplir con nuestro objetivo final. El código para generar esta tabla se muestra a continuación:

```

1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. -- Servidor: localhost
5. -- Tiempo de generación: 24-10-2010 a las 06:19:17
6. -- Versión del servidor: 5.1.36
7. -- Versión de PHP: 5.3.0
8. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
9. -- Base de datos: `test`
10. -----
11. --
12. -- Estructura de tabla para la tabla `grids`
13. CREATE TABLE IF NOT EXISTS `grids` (
14. `id` int(11) NOT NULL AUTO_INCREMENT,
15. `Name` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
16. `Lastname` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
17. `Age` int(3) NOT NULL,
18. `Nationality` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
19. `Sex` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
20. `ZipCode` varchar(15) COLLATE utf8_unicode_ci NOT NULL,
21. `Address` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
22. PRIMARY KEY (`id`)
23.) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT
 =2 ;
24. --
25. -- Volcar la base de datos para la tabla `grids`
26. --
27.
28. INSERT INTO `grids`(`id`, `Name`, `Lastname`, `Age`, `Sex`, `ZipCode`, `Address`) VALUES
29. (1, 'John', 'Smith', 20, 'Male', '50300', 'at home'),
30. (2, 'Nicole', 'Summer', 22, 'Female', '55687', 'somewhere');
```

La base de datos la he llamado “test”, pero puedes usar el nombre que gustes, solamente cuando se haga la conexión mediante PHP recuerda ponerle el nombre correcto.

### Exponer la información

Como sabemos la información es una parte muy importante en una aplicación, en esta ocasión obtendremos esta información desde nuestra base de datos, por lo cual tenemos que hacer la conexión via PHP y un query para poder obtenerla.

```
1. <?php //step 1
2. $connection= mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection) or die("Error loading the DataBase".mysql_error());
4. //step 2
5. $result= mysql_query("SELECT * FROM grids");
6.
7. $data = array();
8. //step 3
9. while($row = mysql_fetch_array($result)){
10. array_push($data, array(
11. "id" => $row["id"],
12. "name" => $row["Name"],
13. "lastname" => $row["Lastname"],
14. "age" => $row["Age"],
15. "sex" => $row["Sex"],
16. "zip" => $row["ZipCode"],
17. "address" => $row["Address"],
18.));
19. }
20. //step 4
21. $fields=array(
22. array("name" => "id"),
23. array("name" => "name","header" =>"Name"),
24. array("name" => "lastname","header" =>"Lastname"),
25. array("name" => "age","header" =>"Age"),
26. array("name" => "sex","header" =>"Sex"),
27. array("name" => "zip","header" =>"ZipCode"),
28. array("name" => "address","header" =>"Address"),
29.);
30. //step 5
31. $metadata = array(
32. "totalProperty" => "results",
33. "successProperty" => "success",
34. "idProperty" => "id",
35. "fields" => $fields,
36. "root" => "data"
37.);
38. //step 6
39. echo json_encode(
40. array(
41. "success" => true,
42. "metaData" => $metadata,
43. "data" => $data
44.));

```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos se crea el query que regresa toda la información de la tabla “grids”.

En el paso tres se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo con la información que contiene la tabla.

Hasta este momento la estructura del documento es conocida, lo interesante viene en el paso cuatro.

En el paso cuatro se crea un nuevo arreglo llamado “fields”, este arreglo lo que contiene son los nombres de los campos que conforman nuestra tabla “grids”.

En el paso cinco se crea el arreglo.

En el paso seis imprimimos la información en formato Json, la respuesta será semejante al siguiente ejemplo:

```
1. {"success":true,"metaData":{"totalProperty":"results","successProperty":"success","idProperty":"id","fields":[{"name":"id"}, {"name":"name","header":"Name"}, {"name":"lastname","header":"Lastname"}, {"name":"age","header":"Age"}, {"name":"sex","header":"Sex"}, {"name":"zip","header":"ZipCode"}, {"name":"address","header":"Address"}], "root":"data"}, "data":[{"id":1,"name":"John","lastname":"Smith","age":20,"sex":Male,"zip":50300,"address":at home}]}]
```

### Encapsulando el código

Ahora pasamos a la parte de JavaScript donde en primer lugar es necesario encapsular el código para evitar problemas en el futuro, así que crearemos un objeto donde alojaremos el código del tutorial.

```
1. Ext.ns("com.quizzpot.tutorial");
2. com.quizzpot.tutorial.Grid= {
3.
4. init : function(){
5. //initial code goes here
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorial.grid.init,com.quizzpot.tutorial.Grid);
```

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### Solicitando la información al servidor

Lo que haremos a continuación es solicitar la información al servidor para poder mostrarla, ya dentro de la función “init” haremos lo siguiente:

```
1. //step 1
2. this.store = new Ext.data.JsonStore({
3. url : "grid.php"
4. });
5. //step 2
6. this.store.on("load",this.createGrid,this);
7. this.store.on("loadexception",this.error,this);
8. this.store.load();
9. },
```

En el paso uno lo que hacemos es crear el “JsonStore” con el cual solicitamos la información al servidor, lo interesante de esto es que el “JsonStore” no recibe ningún parámetro más que “url”, esto es porque si recordamos en el archivo PHP que creamos hace unos instantes contiene los demás parámetros para el “store”.

En el paso dos antes de hacer la petición al servidor tenemos dos eventos “load” y “loadexception”, el primer evento se lleva a cabo cuando el “store” hará la petición al servidor y el segundo será ejecutado si es que algo sale mal mientras el “store” es cargado.

Ambos tienen una función que se encarga de hacer lo necesario para cada uno de estos eventos.

### Función “createGrid”

La función del evento “load” lo que hace es crear el “Grid” con la información que el servidor regresó, esto lo haremos de la siguiente manera:

```
1. createGrid : function(Store,records,options,groups){
2. var cm = [];//step 1
3.
4. Ext.each(this.store.fields.items,function(data){ //step 2
5. if(data.name != "id"){
6. cm.push({header:data.header,dataIndex:data.name,sortable:true}); //step 3
7. }
8. });
9. //.... Seguiremos escribiendo aquí
10. }
```

Creamos la función “createGrid” como normalmente lo hacemos, solo que en esta ocasión le pasamos como parámetros lo que nos regresa el evento “load”.

En el paso uno creamos un arreglo llamado “cm”, el cual lo usaremos para colocar dinámicamente los “headers” de las columnas para nuestro “Grid”.

En el paso dos iteramos la respuesta del servidor, comprobamos en cada iteración que el siguiente campo en la respuesta no sea un “id”. Puedes ver que al arreglo “cm” se le insertan los parámetros necesarios para formar el “header” de una columna, ya que este parámetro le asignamos “data.header” y a “dataIndex” le asignamos “data.name” recordando que esta información está especificada dentro de nuestro archivo PHP.

Con esto tenemos la información necesaria para formar las cabeceras de las columnas del “Grid”, lo siguiente es crearlo y desplegarlo.

```
1. this.grid = new Ext.grid.GridPanel({ //step 1
2. store : this.store,
3. columns : cm,
4. border : false,
5. stripeRows : true
6.);
7.
8. this.win = new Ext.Window({ //step 2
9. title : "Colums",
10. layout : "fit",
11. width : 500,
12. height : 300,
13. items : this.grid
14.);
15. this.win.show();
```

En el paso uno creamos el “Grid”, éste paso ya debería ser conocido para nosotros, pero como pudiste observar generalmente en el atributo “columns” colocamos todas las cabeceras de nuestras columnas, pero en esta ocasión le pasamos el arreglo “cm” que creamos y llenamos anteriormente.

En el paso dos se crea la ventana en la cual mostraremos nuestro “Grid”.

| Name   | Lastname | Age | Sex    | ZipCode | Address   |
|--------|----------|-----|--------|---------|-----------|
| John   | Smith    | 20  | Male   | 50300   | at home   |
| Nicole | Summer   | 22  | Female | 55687   | somewhere |

Colocando el Grid

### Función “error”

Esta es la función del evento “loadexception”, lo único que haré en esta función es mandar la información que se generó, con el fin de poder buscar donde está el error y el porqué no se puede mostrar nuestra información.

Esta función tiene la siguiente estructura:

1. error :function(){
2. console.debug(arguments)
3. }

Como mencionamos hace uno instantes lo único que hace esta función es mandar a consola la información por la cual se pudo originar el error.

### Conclusión

En esta ocasión vimos que podemos crear las cabeceras de nuestro “Grid” de una manera más dinámica, esto resulta efectivo en grandes proyectos, ya que esto nos ayuda a simplificar y automatizar la creación de los “grids”.

## Edición de tablas

En ocasiones es conveniente poder editar la información directamente en una tabla, de esta manera evitamos abrir un formulario con la información en cada campo.

### Editar las celdas de un grid

El día de hoy aprenderemos como editar las celdas de un Grid de una manera muy sencilla, realmente te sorprenderás cuán fácil y rápido es.

Ya sabemos como crear un Grid utilizando una llamada Ajax para llenar el contenido, lo que vamos hacer hoy es agregar la funcionalidad de poder editar el contenido de una celda, para ello cuando el usuario de doble clic sobre una celda aparecerá una caja de texto donde podrá capturar el nuevo contenido.

#### Demostración

Si quieres ver el ejemplo que haremos al final de este tutorial te invito a probar la demostración que he preparado.

|    | City            | Visits | Page/Visits | Average Time |
|----|-----------------|--------|-------------|--------------|
| 1  | Mexico city     | 684    | 4.11        | 13:03:58     |
| 2  | La Victoria     | 443    | 4.39        | 13:03:58     |
| 3  | Madrid - España | 380    | 3.12        | 13:03:58     |
| 4  | Providencia     | 204    | 3.83        | 13:03:58     |
| 5  | Bogota          | 204    | 3.26        | 13:03:58     |
| 6  | Puerto Madero   | 192    | 3.56        | 13:03:58     |
| 7  | Monterrey       | 174    | 3.9         | 13:03:58     |
| 8  | Barcelona       | 145    | 3.28        | 13:03:58     |
| 9  | Caracas         | 132    | 4.55        | 13:03:58     |
| 10 | Rosario         | 116    | 2.44        | 13:03:58     |
| 11 | Oaxaca          | 108    | 1.73        | 13:03:58     |

Demostración del ejemplo

#### Material de apoyo

Para continuar es necesario descargar el material de apoyo que usaremos para este ejemplo, luego de hacerlo asegúrate de copiarlo al servidor web donde hemos estado trabajando, ahí crea una carpeta que se llame "editorgrid" y pega los tres archivos dentro de esta. Asegúrate de que las rutas a la librería de Ext JS sean las correctas en el "html" que has descargado.

#### El servidor

Antes de seguir adelante quiero mostrar el contenido del archivo "editorgrid.php" (viene en el material de apoyo).

```
1. <?php
2. header("Content-Type: text/plain");
3.
4. $data = array(
5. 'success'=>true,
6. 'total'=>11,
7. 'data'=>array(
8. array('city'=>'Mexico city','visits'=>684,'pageVisits'=>4.11,'averageTime'=>'00:06:53','time'
=>date('U')),
```

```

9. array('city'=>'La Victoria','visits'=>443,'pageVisits'=>4.39,'averageTime'=>'00:07:28','time'=>date('U')),
10. array('city'=>'Madrid','visits'=>380,'pageVisits'=>3.11,'averageTime'=>'00:05:22','time'=>date('U')),
11. array('city'=>'Providencia','visits'=>204,'pageVisits'=>3.83,'averageTime'=>'00:08:20','time'=>date('U')),
12. array('city'=>'Bogota','visits'=>204,'pageVisits'=>3.26,'averageTime'=>'00:04:57','time'=>date('U')),
13. array('city'=>'Puerto Madero','visits'=>192,'pageVisits'=>3.56,'averageTime'=>'00:05:07','time'=>date('U')),
14. array('city'=>'Monterrey','visits'=>174,'pageVisits'=>3.90,'averageTime'=>'00:06:06','time'=>date('U')),
15. array('city'=>'Barcelona','visits'=>145,'pageVisits'=>3.28,'averageTime'=>'00:05:39','time'=>date('U')),
16. array('city'=>'Caracas','visits'=>132,'pageVisits'=>4.55,'averageTime'=>'00:06:27','time'=>date('U')),
17. array('city'=>'Rosario','visits'=>116,'pageVisits'=>2.44,'averageTime'=>'00:04:30','time'=>date('U')),
18. array('city'=>'Oaxaca','visits'=>108,'pageVisits'=>1.73,'averageTime'=>'00:02:37','time'=>date('U'))
19.)
20.);
21.
22. echo json_encode($data);
23. ?>

```

En el código anterior la información está contenida en un arreglo, quiero aclarar que puede estar en una base de datos o en cualquier otro lugar, pero lo hice de esta forma para hacer más sencillo el tutorial.

Lo que debemos tomar en cuenta del código anterior es la manera en que se esta imprimiendo el JSON, pues ese formato debemos definirlo en el store que crearemos a continuación.

### Definiendo el “JsonStore”

Vamos a utilizar un JsonStore para almacenar la información que el servidor nos provee y poder manipularla en el grid.

```

1. var store = new Ext.data.JsonStore({
2. url: 'editorgrid.php',
3. root: 'data',
4. fields: ['city',{name:'visits',type:'float'},{name:'pageVisits',type:'float'},{name:'averageTime',type:'date',dateFormat: 'H:i:s'},
5. {name:'time',type:'date', dateFormat: 'U'}]
6. });

```

El código anterior debe ser muy familiar para nosotros a estas alturas del curso, pues lo hemos utilizado muchas veces en los tutoriales anteriores e inclusive dedicamos un tutorial al respecto.

### Crear el Grid

Ahora vamos a crear el grid que mostrará la información, para esto vamos a utilizar el componente “Ext.grid.EditorGridPanel” el cual nos permite editar las celdas, la configuración del componente será exactamente igual a la configuración de un Grid normal.

```

1. var grid = new Ext.grid.EditorGridPanel({
2. store: store,
3. columns: [
4. new Ext.grid.RowNumberer(),
5. {header:'City', dataIndex:'city', sortable: true},
6. {header:'Visits', dataIndex:'visits', sortable: true},
7. {header:'Page/Visits', dataIndex:'pageVisits', sortable: true},

```

```

8. {header:'Average Time', dataIndex:'time', width:150, sortable: true, renderer: Ext.util.Format.dateRenderer('H:i:s')}
9.],
10. border: false,
11. stripeRows: true
12. });

```

Al ver el código anterior nos damos cuenta que no hay diferencia alguna en las configuraciones con respecto a un "Grid", hasta aquí no debe haber ninguna duda, si es así te recomiendo leer el tutorial donde hablamos sobre el Grid.

### **Mostrar la ventana contenedora**

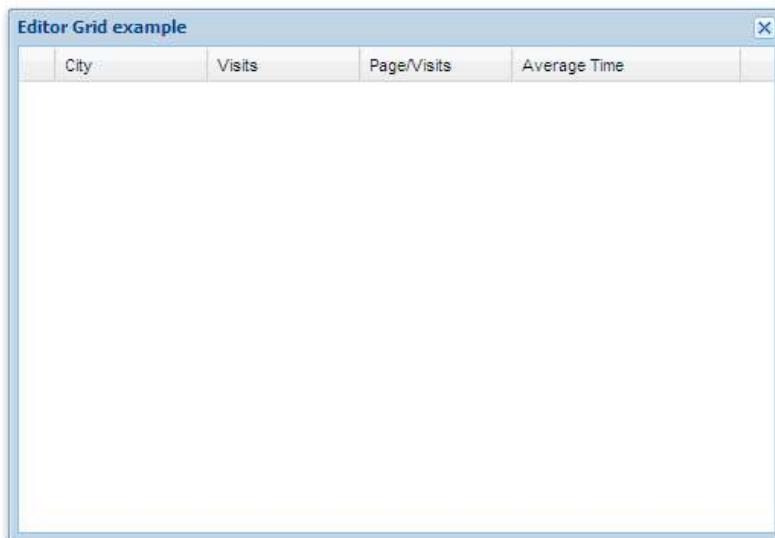
Hasta ahora no hemos visto nada en la pantalla del navegador, esto es porque en ningún momento hemos "renderizado" el grid, así que vamos a utilizar una ventana que contenga al grid anterior.

```

1. var win = new Ext.Window({
2. title: 'Editor Grid example',
3. layout: 'fit',
4. width: 510,
5. height: 350,
6. items: grid //Le asignamos el grid
7. });
8.
9. win.show(); //mostramos la ventana y su contenido

```

Si todo ha salido bien deberíamos ver algo semejante a la siguiente imagen.



Ventana con Grid vacío

### **Cargar la información en el "Grid"**

Pero, ¿Dónde está la información? Si te das cuenta todavía no hemos realizado la llamada Ajax al servidor solicitando la información que tendrá el Grid, así que vamos hacerlo de la siguiente manera.

1. //Solicitar la información al servidor
2. store.load();

Ejecutando el método "load" del store recogemos la información, esto ya lo hemos estudiado anteriormente, pero es bueno recordarlo para aquellos que apenas comienzan.

|    | City          | Visits | Page/Visits | Average Time |  |
|----|---------------|--------|-------------|--------------|--|
| 1  | Mexico city   | 684    | 4.11        | 13:19:39     |  |
| 2  | La Victoria   | 443    | 4.39        | 13:19:39     |  |
| 3  | Madrid        | 380    | 3.11        | 13:19:39     |  |
| 4  | Providencia   | 204    | 3.83        | 13:19:39     |  |
| 5  | Bogota        | 204    | 3.26        | 13:19:39     |  |
| 6  | Puerto Madero | 192    | 3.56        | 13:19:39     |  |
| 7  | Monterrey     | 174    | 3.9         | 13:19:39     |  |
| 8  | Barcelona     | 145    | 3.28        | 13:19:39     |  |
| 9  | Caracas       | 132    | 4.55        | 13:19:39     |  |
| 10 | Rosario       | 116    | 2.44        | 13:19:39     |  |
| 11 | Oaxaca        | 108    | 1.73        | 13:19:39     |  |

Grid con información cargada con Ajax

Es interesante notar que al dar clic sobre una celda ésta se selecciona, en un grid normal (con la misma configuración que tenemos en este ejemplo) se debería seleccionar toda la fila por defecto.

#### Mostrar una caja de texto al dar doble clic

Ahora viene lo interesante de este tutorial, vamos hacer que al dar doble clic sobre una celda aparezca una caja de texto donde podamos escribir y modificar la información.

Primero necesitamos crear la caja de texto de la siguiente manera:

1. //Creamos el texfield antes de crear el grid
2. var textField = new Ext.form.TextField();

Después debemos asignársela a la columna donde deseamos que aparezca al dar doble clic sobre alguna de sus celdas, para esto utilizamos la propiedad "editor" en la columna.

1. //Asignamos el texfield anterior a la columnas deseada
2. {header:'City', dataIndex:'city', sortable: true, editor:textField},

Esto es suficiente para lograr nuestro objetivo, actualiza tu explorador y da doble clic sobre alguna celda de la columna "city" y observa lo que sucede.

|    | City            | Visits | Page/Visits | Average Time |  |
|----|-----------------|--------|-------------|--------------|--|
| 1  | Mexico city     | 684    | 4.11        | 13:28:12     |  |
| 2  | La Victoria     | 443    | 4.39        | 13:28:12     |  |
| 3  | Madrid - España | 380    | 3.11        | 13:28:12     |  |
| 4  | Providencia     | 204    | 3.83        | 13:28:12     |  |
| 5  | Bogota          | 204    | 3.26        | 13:28:12     |  |
| 6  | Puerto Madero   | 192    | 3.56        | 13:28:12     |  |
| 7  | Monterrey       | 174    | 3.9         | 13:28:12     |  |
| 8  | Barcelona       | 145    | 3.28        | 13:28:12     |  |
| 9  | Caracas         | 132    | 4.55        | 13:28:12     |  |
| 10 | Rosario         | 116    | 2.44        | 13:28:12     |  |
| 11 | Oaxaca          | 108    | 1.73        | 13:28:12     |  |

Editar una celda al dar doble clic

## Filtrar solamente números

Podemos también asignarle a la propiedad “editor” algún otro componente de captura de información, en este caso vamos a crear un “NumberField” para capturar solamente números, adicionalmente vamos a configurarlos para forzar al usuario a introducir algún valor y que no deje vacío el campo.

```
1. var numberField = new Ext.form.NumberField({allowBlank:false});
```

Lo siguiente es asignárselos a las columnas que necesitemos.

1. //reutilizamos el componente en varias columnas
2. {header:'Visits', dataIndex:'visits', sortable: true, editor:numberField},
3. {header:'Page/Visits', dataIndex:'pageVisits', sortable: true, editor:numberField},

Al hacer esto vemos como ahora valida que el usuario escriba algo en la caja de texto, además de que solamente permite capturar números.

The screenshot shows a window titled "Editor Grid example". Inside is a grid with four columns: "City", "Visits", "Page/Visits", and "Average Time". The "Visits" column contains numerical values. Row 6, which corresponds to "Puerto Madero", has its "Visits" value, "3.56", highlighted with a red rectangular selection box. The other rows show values like 684, 443, 380, etc.

|    | City          | Visits | Page/Visits | Average Time |
|----|---------------|--------|-------------|--------------|
| 1  | Mexico city   | 684    | 4.11        | 13:33:27     |
| 2  | La Victoria   | 443    | 4.39        | 13:33:27     |
| 3  | Madrid        | 380    | 3.11        | 13:33:27     |
| 4  | Providencia   | 204    | 3.83        | 13:33:27     |
| 5  | Bogota        | 204    | 3.26        | 13:33:27     |
| 6  | Puerto Madero | 3.56   | 13:33:27    |              |
| 7  | Monterrey     | 174    | 3.9         | 13:33:27     |
| 8  | Barcelona     | 145    | 3.28        | 13:33:27     |
| 9  | Caracas       | 132    | 4.55        | 13:33:27     |
| 10 | Rosario       | 116    | 2.44        | 13:33:27     |
| 11 | Oaxaca        | 108    | 1.73        | 13:33:27     |

Filtrar solo números

## Campos sucios

Cuando modificamos la información original contenida en las celdas aparece un ícono en la parte superior izquierda de la celda, este ícono nos indica que la celda ha sido modificada, Ext JS denomina a estos registros como “Dirty” que traducido al español significa “Sucio”, esta funcionalidad es propia del componente y no debemos hacer nada para que aparezca.

The screenshot shows a window titled "Editor Grid example". Inside is a grid with four columns: "City", "Visits", "Page/Visits", and "Average Time". The first column, "City", contains names with some having suffixes like "- España", "- Colombia", or "- México". Most cells in this column have a small red triangle icon in the top-left corner, indicating they are dirty. The "Visits" column contains numerical values. Row 5, which corresponds to "Bogota - Colombia", has its "Visits" value, "209", highlighted with a blue rectangular selection box. The other rows show values like 684, 443, 380, etc.

|    | City               | Visits | Page/Visits | Average Time |
|----|--------------------|--------|-------------|--------------|
| 1  | Mexico city        | 684    | 4.11        | 13:33:27     |
| 2  | La Victoria        | 443    | 4.39        | 13:33:27     |
| 3  | Madrid - España    | 380    | 3.11        | 13:33:27     |
| 4  | Providencia        | 204    | 3.83        | 13:33:27     |
| 5  | Bogota - Colombia  | 209    | 3.26        | 13:33:27     |
| 6  | Puerto Madero      | 192    | 3.56        | 13:33:27     |
| 7  | Monterrey - México | 174    | 3.9         | 13:33:27     |
| 8  | Barcelona          | 145    | 3.28        | 13:33:27     |
| 9  | Caracas            | 132    | 4.55        | 13:33:27     |
| 10 | Rosario            | 116    | 2.43        | 13:33:27     |
| 11 | Oaxaca             | 108    | 1.73        | 13:33:27     |

Campos “Dirty”

## Navegación con el teclado

Una de las funcionalidades que vale la pena mencionar es que podemos movernos entre las celdas usando el teclado, simplemente utiliza las flechas del teclado para moverte y si estás editando una celda presiona la tecla "Tab" para pasar a editar la siguiente celda, o bien presiona "Shift + Tab" para pasar a editar la celda anterior, para entrar a editar una celda presiona "Enter" y para salir del modo edición puede presionar "Enter" o "Esc".

|    | City               | Visits | Page/Visits | Average Time |
|----|--------------------|--------|-------------|--------------|
| 1  | Mexico city        | 684    | 4.11        | 13:33:27     |
| 2  | La Victoria        | 443    | 4.39        | 13:33:27     |
| 3  | Madrid - España    | 380    | 3.11        | 13:33:27     |
| 4  | Providencia        | 204    | 3.83        | 13:33:27     |
| 5  | Bogota - Colombia  | 209    | 3.26        | 13:33:27     |
| 6  | Puerto Madero      | 192    | 3.56        | 13:33:27     |
| 7  | Monterrey - México | 174    | 3.9         | 13:33:27     |
| 8  | Barcelona          | 145    | 3.28        | 13:33:27     |
| 9  | Caracas            | 132    | 4.55        | 13:33:27     |
| 10 | Rosario            | 116    | 2.43        | 13:33:27     |
| 11 | Oaxaca             | 108    | 1.73        | 13:33:27     |

Usando el teclado para movernos en el Grid

## Conclusiones

En este tema vimos como editar las celdas utilizando cajas de texto, realmente es algo sumamente sencillo; en futuros tutoriales mostraré como utilizar combos y algunos otros componentes, además de cómo guardar la información capturada ya que no nos sirve de nada editarla si no la podemos guardar.

## Editar una celda mediante un Combobox

Normalmente tenemos relaciones en nuestras tablas de la base de datos y una de las mejores maneras para mostrárselas al usuario es mediante un Combobox que despliegue los registros de la tabla que necesitamos relacionar.

Ya vimos como editar el contenido de las celdas mediante una caja de texto, en este tutorial vamos a crear una Grid que nos permita editar el contenido de una celda utilizando un Combobox para seleccionar una de varias opciones.

## Demostración

He preparado una demostración de lo que haremos al final del tutorial, te invito a que la pruebes para que tengas en mente el objetivo final.

| USA Weekend Box-Office Summary |           |                                        |      |         |          |  |
|--------------------------------|-----------|----------------------------------------|------|---------|----------|--|
|                                | Genre     | Title                                  | Year | Weekend | All Time |  |
| 1                              | Animation | G-Force                                | 2009 | \$32.2M | \$32.2M  |  |
| 2                              | Fantasy   | Harry Potter and the Half-Blood Prince | 2009 | \$30M   | \$222M   |  |
| 3                              | Comedy    | The Ugly Truth                         | 2009 | \$27M   | \$27M    |  |
| 4                              | Drama     | Orphan                                 | 2009 | \$12.8M | \$12.8M  |  |
| 5                              | Animation | Ice Age: Dawn of the Dinosaurs         | 2009 | \$8.2M  | \$171M   |  |
| 6                              | Action    | Transformers: Revenge of the Fallen    | 2009 | \$8M    | \$379M   |  |
| 7                              | Animation | The Hangover                           | 2009 | \$6.46M | \$247M   |  |
| 8                              | Comedy    | The Proposal                           | 2009 | \$6.42M | \$140M   |  |
| 9                              | Drama     | Public Enemies                         | 2009 | \$4.17M | \$88.1M  |  |
| 10                             | Fantasy   | Bruno                                  | 2009 | \$2.72M | \$56.5M  |  |
|                                | Horror    |                                        |      |         |          |  |
|                                | Musical   |                                        |      |         |          |  |
|                                | Romance   |                                        |      |         |          |  |

Demostración del tutorial

### Material de apoyo

Antes de continuar te recomiendo descargar el material de apoyo que he preparado, recuerda que se tiene que copiar dentro de un servidor Web, en una carpeta que se llama “editorgrid” donde estamos trabajando.

Puedes ver que en el material de apoyo vienen dos archivos PHP, esto es para simular dos diferentes “controllers” que nos regresan información diferente, uno de ellos (editorgrid-combo.php) nos regresa la información que irá en el Grid y el otro (editorgrid-combo-cmb.php) la información con la que llenaremos las opciones del combo.

### Información del Grid

Lo primero que debemos hacer es definir la manera en que recibiremos la información desde nuestro “controller” en este caso “editorgrid-combo.php”.

```

1. <?php
2. header("Content-Type: text/plain");
3.
4. $data = array(
5. array('genre'=>2,'title'=>'G-Force','year'=>2009,'weekend'=>32.2,'allTime'=>32.2),
6. array('genre'=>5,'title'=>'Harry Potter and the Half-
 Blood Prince','year'=>2009,'weekend'=>30,'allTime'=>222),
7. array('genre'=>3,'title'=>'The Ugly Truth','year'=>2009,'weekend'=>27,'allTime'=>27),
8. array('genre'=>4,'title'=>'Orphan','year'=>2009,'weekend'=>12.8,'allTime'=>12.8),
9. array('genre'=>2,'title'=>'Ice Age: Dawn of the Dinosaurs ','year'=>2009,'weekend'=>8.2,'all
 Time'=>171),
10. array('genre'=>1,'title'=>'Transformers: Revenge of the Fallen','year'=>2009,'weekend'=>8,
 'allTime'=>379),
11. array('genre'=>3,'title'=>'The Hangover','year'=>2009,'weekend'=>6.46,'allTime'=>247),
12. array('genre'=>3,'title'=>'The Proposal','year'=>2009,'weekend'=>6.42,'allTime'=>140),
13. array('genre'=>1,'title'=>'Public Enemies','year'=>2009,'weekend'=>4.17,'allTime'=>88.1),
14. array('genre'=>8,'title'=>'Bruno','year'=>2009,'weekend'=>2.72,'allTime'=>56.5)
15.
16.);
17.
18. echo json_encode(array(
19. 'success'=>true,
20. 'total'=>count($data),
21. 'data'=> $data
22.));
23. ?>

```

He decidido utilizar JSON para el intercambio de información, además se definieron los campos que desplegaremos en el Grid.

### Crear el Grid editable

Con el código anterior en mente vamos a crear el Grid Editable, no me voy a detener a explicar este paso pues ya lo hice en repetidas ocasiones en temas pasados.

```
1. var store = new Ext.data.JsonStore({ // 1
2. url: 'editorgrid-combo.php',
3. root: 'data',
4. totalProperty: 'total',
5. fields: ['genre','title','year','weekend','allTime']
6. });
7. store.load(); //2
8.
9. var grid = new Ext.grid.EditorGridPanel({ //3
10. store: store,
11. columns: [
12. new Ext.grid.RowNumberer(),
13. {header:'Genre', dataIndex:'genre', width:100, sortable: true},
14. {header:'Title', dataIndex:'title', width:200, sortable: true},
15. {header:'Year', dataIndex:'year', width:60, sortable: true},
16. {header:'Weekend', dataIndex:'weekend', width:55, sortable: true},
17. {header:'All Time', dataIndex:'allTime', width:55, sortable: true}
18.],
19. border: false,
20. stripeRows: true
21. });
```

Primero creamos el “store” donde definimos los campos que contendrán los registros, además configuramos la “url” de donde proviene la información (1), después hacemos la petición al servidor mediante el método “load” (2) y por último creamos el Grid Editable con las columnas que necesitamos.

### Crear la ventana contenedora

Hasta este punto no hemos visto nada en pantalla porque no se ha renderizado el Grid, vamos a usar una ventana para desplegarlos.

```
1. var win = new Ext.Window({
2. title: 'USA Weekend Box-Office Summary',
3. layout: 'fit',
4. width: 520,
5. height:300,
6. items: grid
7. });
8. win.show();
```

| USA Weekend Box-Office Summary |       |                                        |      |         |          |  |
|--------------------------------|-------|----------------------------------------|------|---------|----------|--|
|                                | Genre | Title                                  | Year | Weekend | All Time |  |
| 1                              | 2     | G-Force                                | 2009 | 32.2    | 32.2     |  |
| 2                              | 5     | Harry Potter and the Half-Blood Prince | 2009 | 30      | 222      |  |
| 3                              | 3     | The Ugly Truth                         | 2009 | 27      | 27       |  |
| 4                              | 4     | Orphan                                 | 2009 | 12.8    | 12.8     |  |
| 5                              | 2     | Ice Age: Dawn of the Dinosaurs         | 2009 | 8.2     | 171      |  |
| 6                              | 1     | Transformers: Revenge of the Fallen    | 2009 | 8       | 379      |  |
| 7                              | 3     | The Hangover                           | 2009 | 6.46    | 247      |  |
| 8                              | 3     | The Proposal                           | 2009 | 6.42    | 140      |  |
| 9                              | 1     | Public Enemies                         | 2009 | 4.17    | 88.1     |  |
| 10                             | 8     | Bruno                                  | 2009 | 2.72    | 56.5     |  |

Grid editable

Lo que hemos hecho debe estar perfectamente claro, pero si tienes alguna duda puedes ir a repasar los tutoriales anteriores donde hablo mas a detalle de lo sucedido.

### Crear el combo para seleccionar los años

Vamos a crear nuestro primer combo, éste será muy sencillo y lo usaremos para permitir que el usuario pueda seleccionar un año.

```

1. var comboYear = {
2. xtype: 'combo',
3. triggerAction : 'all',
4. store: [2009,2008,2007,2006]
5. }
```

Quiero hacer notar que el código anterior es solamente la configuración de un combo que desplegará cuatro años anteriores (2009, 2008, 2007, 2006), he decidido hacerlo estático para fines didácticos.

Lo siguiente es asignar este combo a la columna donde necesitemos desplegarlo en el Grid Editable, esto lo hacemos mediante la propiedad “editor”.

```
1. {header:'Year', dataIndex:'year', width:60, sortable: true, editor:comboYear},
```

Ahora actualizamos el navegador y veremos algo como la siguiente imagen.

|    | Genre | Title                                  | Year | Weekend | All Time |
|----|-------|----------------------------------------|------|---------|----------|
| 1  | 2     | G-Force                                | 2009 | 32.2    | 32.2     |
| 2  | 5     | Harry Potter and the Half-Blood Prince | 2009 | 30      | 222      |
| 3  | 3     | The Ugly Truth                         | 2009 | 27      | 27       |
| 4  | 4     | Orphan                                 | 2009 | 12.8    | 12.8     |
| 5  | 2     | Ice Age: Dawn of the Dinosaurs         | 2009 | 8.2     | 171      |
| 6  | 1     | Transformers: Revenge of the Fallen    | 2009 | 8       | 379      |
| 7  | 3     | The Hangover                           | 2009 | 6.46    | 247      |
| 8  | 3     | The Proposal                           | 2009 | 6.42    | 140      |
| 9  | 1     | Public Enemies                         | 2009 | 5.17    | 88.1     |
| 10 | 8     | Brüno                                  | 2008 | 5.72    | 56.5     |
|    |       |                                        | 2007 |         |          |
|    |       |                                        | 2006 |         |          |

Grid editable con combo estático

Como puedes ver se ha desplegado un combo en el cual podemos seleccionar un año diferente, esto ha sido realmente sencillo, en tan solo unas pocas líneas de código lo hemos logrado.

### Crear un combo dinámico

El combo que hicimos anteriormente era muy sencillo, en el mundo real no todo es tan sencillo, lo que vamos hacer ahora es crear un combo cuya información provenga de una tabla en la base de datos que se llame "Genres" que esta relacionada de uno a muchos (one-to-many) con otra que se llama "Movies".

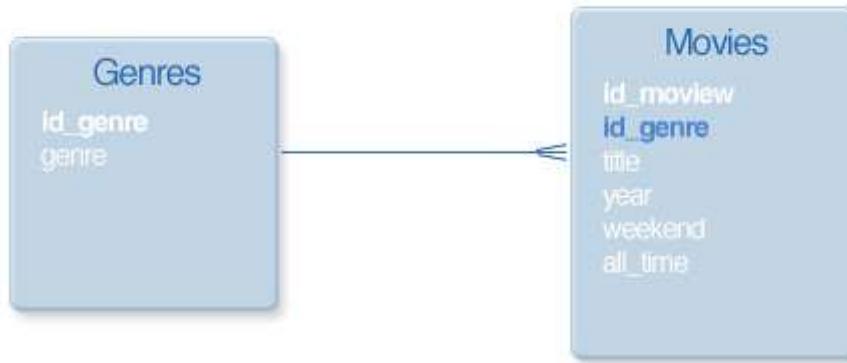


Diagrama de base de datos (relación “one-to-many”)

Dada la imagen anterior podemos decir que “Un genero tiene muchas películas”.

Necesitamos sacar la información de la tabla y exponerla mediante un “controller”, para este ejemplo he creado el siguiente archivo PHP encargado de mostrar la información para el combo (editorgrid-combo-cmb.php), es importante mencionar que para fines didácticos la información está en un arreglo, pero supongamos que esta información viene de la tabla “Genres” en la base de datos:

1. <?php
2. header("Content-Type: text/plain");
- 3.
4. \$data = array(
5.     array('value'=>1,'label'=>'Action'),
6.     array('value'=>2,'label'=>'Animation'),

```

7. array('value'=>3,'label'=>'Comedy'),
8. array('value'=>4,'label'=>'Drama'),
9. array('value'=>5,'label'=>'Fantasy'),
10. array('value'=>6,'label'=>'Horror'),
11. array('value'=>7,'label'=>'Musical'),
12. array('value'=>8,'label'=>'Romance')
13.);
14.
15. echo json_encode(array(
16. 'records'=>$data
17.));
18. ?>

```

Con el código anterior en mente vamos a crear el combo que despliegue la información expuesta de la tabla “Genres” y asignárselo a la columna correspondiente en el Grid.

```

1. this.store = new Ext.data.JsonStore({
2. root : 'records',
3. fields : ['value','label'],
4. url: 'editorgrid-combo-cmb.php' //different controller
5. });
6. this.store.load();

```

Primero creamos el store para el combo, puedes ver que la “url” es diferente al store del grid, esto es porque en el “controller” estamos consultamos la tabla “Genre” para llenar el Combobox, además estamos creando una variable “this.store” para tener una referencia al store que usaremos más adelante.

```

1. var comboGenre = new Ext.form.ComboBox({
2. triggerAction : 'all',
3. displayField : 'label',
4. valueField : 'value',
5. store : this.store
6. });

```

Por ultimo agregamos el combo a la columna donde se desplegará por medio de la propiedad “editor”.

```
1. {header:'Genre', dataIndex:'genre',width:100,sortable: true,editor:comboGenre},
```

Guardemos los cambios y actualicemos la página en el explorador y veremos algo semejante a la siguiente imagen.

| USA Weekend Box-Office Summary |           |                                        |      |         |          |
|--------------------------------|-----------|----------------------------------------|------|---------|----------|
|                                | Genre     | Title                                  | Year | Weekend | All Time |
| 1                              | 2         | G-Force                                | 2009 | 32.2    | 32.2     |
| 2                              | 5         | Harry Potter and the Half-Blood Prince | 2009 | 30      | 222      |
| 3                              | 3         | The Ugly Truth                         | 2009 | 27      | 27       |
| 4                              | Musical   | Orphan                                 | 2009 | 12.8    | 12.8     |
| 5                              | Action    | Ice Age: Dawn of the Dinosaurs         | 2009 | 8.2     | 171      |
| 6                              | Animation | Transformers: Revenge of the Fallen    | 2009 | 8       | 379      |
| 7                              | Comedy    | The Hangover                           | 2009 | 6.46    | 247      |
| 8                              | Drama     | The Proposal                           | 2009 | 6.42    | 140      |
| 9                              | Fantasy   | Public Enemies                         | 2009 | 4.17    | 88.1     |
| 10                             | Horror    | Brüno                                  | 2009 | 2.72    | 56.5     |
|                                | Romance   |                                        |      |         |          |

Combobox dinámico en el grid editable

Puedes ver que al dar doble clic sobre alguna de las celdas de la columna “genre” se despliega el combo cuya información ha sido cargada dinámicamente desde una tabla en la base de datos y nos sirve para hacer la relación “one to many”.

### Desplegar el género y no el identificador

Hasta ahora hemos desplegado la información de la tabla “Movies” en el Grid, pero si has notado esta apareciendo el “id\_genre” (que es la llave foránea) en la primera columna, esto no es muy intuitivo y no nos sirve de nada, por lo tanto lo que debemos hacer es reemplazar ese identificador por su respectivo género.

Lo que haremos para solucionar este problema es utilizar la propiedad “renderer” para cambiar el “id” por el texto que necesitamos.

```
1. {header:'Genre', dataIndex:'genre',width:100,sortable: true, editor:comboGenre, renderer:this.genre},
```

Ahora necesitamos crear la función “this.genre” que especificamos en el código anterior.

```
1. com.quizzpot.tutorial.ComboGridTutorial = {
2. init: function(){
3. //código removido
4. },
5. //esta función es la que se ejecutará antes de
6. //que se renderize el contenido de la celda
7. genre: function(value){
8.
9. },
10.
11. money: function(id){
12. //código removido
13. }
14. }
```

Ahora vamos a cambiar el valor por defecto (el “id”) por el respectivo texto, para esto vamos a utilizar el store del combo que contiene la información que necesitamos ¿recuerdan que les comenté que lo usaríamos más adelante? Pues aquí lo usaremos, por eso es que cuando lo definí hice que fuera una propiedad del objeto, de esta manera podemos usarlo dentro de esta función de la siguiente manera:

```
1. genre: function(id){
2. var index = this.store.find('value',id);
3. if(index>-1){
4. var record = this.store.getAt(index);
5. return record.get('label');
6. }
7. return value;
8. }
```

En el código anterior primero busco donde se encuentra el registro cuyo campo “value” (que es el “id\_genre” en la tabla “Genres” de la base de datos) es igual al “id” que estoy recibiendo (la “llave foránea” que se encuentra en la tabla “Movies”), si encontró un registro (debería encontrarlo siempre, de lo contrario nuestra base de datos no es consistente) lo saca del store para poder obtener el “label” del “id” correspondiente.

Ahora si, todo debería funcionar correctamente, actualiza tu explorador y verás algo como lo siguiente.

The screenshot shows the Firebug extension's interface. At the top, there are tabs for Console, HTML, CSS, Script, DOM, and Net. The Console tab is active and displays two log entries:

- [+] GET http://localhost:8081/course/editorgrid/editorgrid-combo.php?\_dc=1251477445427
- [+] GET http://localhost:8081/course/editorgrid/editorgrid-combo-cmb.php?\_dc=1251477445

Below these, an error message is shown in red text: "this.store is undefined". A yellow callout box highlights the line of code causing the error: "var index = this.store.find('value', id); \r\n".

At the bottom of the console, there is a URL bar with "http://localhost:8081/course/editorgrid/editorgrid-combo.html#" and the text "Error en el scope" below it.

### Cambiando el “scope”

Ops!! ¿Que pasó? Lo que ha sucedido es que la función que acabamos de escribir se está ejecutando en un “scope” diferente, por lo tanto al hacer referencia a la variable “this.store” nos dice que no existe, para solucionar este problema necesitamos especificarle el “scope” que usará al momento de asignársela a la columna, ya sea mediante la propiedad “scope” o utilizando el método “createDelegate”.

1. {
2. header:'Genre', dataIndex:'genre',width:100,sortable: true,
3. editor:comboGenre,renderer:this.genre,
4. scope: this //definimos el scope para la función “this.genre”
5. },

O también funcionaría así:

1. {
2. header:'Genre', dataIndex:'genre',width:100,sortable: true,
3. editor:comboGenre,
4. renderer:this.genre.createDelegate(this)//definimos el scope de la función
5. },

Ahora si probemos nuevamente el ejemplo en el navegador y debemos ver algo semejante a la siguiente imagen:

| USA Weekend Box-Office Summary |           |                                        |      |         |          |
|--------------------------------|-----------|----------------------------------------|------|---------|----------|
|                                | Genre     | Title                                  | Year | Weekend | All Time |
| 1                              | Animation | G-Force                                | 2009 | 32.2    | 32.2     |
| 2                              | Fantasy   | Harry Potter and the Half-Blood Prince | 2009 | 30      | 222      |
| 3                              | Comedy    | The Ugly Truth                         | 2009 | 27      | 27       |
| 4                              | Drama     | Orphan                                 | 2009 | 12.8    | 12.8     |
| 5                              | Animation | Ice Age: Dawn of the Dinosaurs         | 2009 | 8.2     | 171      |
| 6                              | Action    | Transformers: Revenge of the Fallen    | 2009 | 8       | 379      |
| 7                              | Comedy    | The Hangover                           | 2009 | 6.46    | 247      |
| 8                              | Comedy    | The Proposal                           | 2009 | 6.42    | 140      |
| 9                              | Action    | Public Enemies                         | 2009 | 4.17    | 88.1     |
| 10                             | Romance   | Brüno                                  | 2009 | 2.72    | 56.5     |

Cambiando el ID por un Texto

### **Asegurarse que siempre exista contenido en el store del combo**

Por último debemos asegurarnos que se cargue primero el contenido del combo y luego el contenido del grid, debemos hacerlo así porque estamos usando el store del combo para renderizar el contenido del grid.

1. //una vez que se cargue el store del combo...
2. this.store.on('load',function(){
3.     store.load();//...cargamos el store del grid
4. });

Solamente necesitamos agregar un “listener” al evento “load” del store del combo para que inmediatamente después de que se cargue solicitemos la información del Grid, así evitamos que aparezcan los identificadores.

### **Conclusiones**

El tutorial de hoy es muy importante, aprendimos como crear combos estáticos y asignárselos a las columnas para ser editadas, pero también aprendimos ha utilizar los combos para crear relaciones entre dos tablas de “one-to-many”, esto es muy común cuando desarrollamos sistemas, también hablamos un poco acerca del “scope”. De tarea les dejo que le agreguen campos de texto a las celdas que hacen falta y que le agreguen el “renderer: this.money” a las ultimas dos celdas donde se muestran cantidades de dinero.

## **Guardar los cambios del Grid editable usando Ajax**

En el tutorial de hoy veremos como guardar la información de un Grid editable utilizando el componente Ajax que viene con el Framework, también mostraré como agregar nuevos registros al Grid y guardarlos en el servidor.

En tutoriales anteriores hemos visto como crear un Grid con capacidad de editar las celdas mediante una caja de texto o un Combobox, pero realmente eso no nos sirve de mucho si no guardamos esos cambios en nuestro servidor, el día de hoy voy a mostrar la manera tradicional de guardar estos cambios.

Es importante mencionar que en Ext 3 hay una manera más sencilla de hacer esto (utilizando un “Writer”), pero quiero mostrar el método “tradicional” que usábamos en la versión de Ext 2 porque creo que es importante para tener una compresión clara de lo que sucede ya que en futuros tutoriales mostraré una forma más sencilla.

### **Material de apoyo**

Para comenzar es necesario descargar el material de apoyo que consta de un archivo HTML donde únicamente se importa el Framework y hay clases “CSS” definidas, también encontrarás un archivo “JS” donde he definido el “namespace” y además algunos métodos que implementaremos en este tutorial, viene también un archivo “PHP” el cual nos servirá para simular el guardado y actualizado de información y por último una carpeta “icons” que contiene unas imágenes que mostraremos en los botones.

### **Demostración**

He preparado una demostración de lo que haremos al término de este tutorial, te invito a que la pruebes por ti mismo.

**Editor Grid example**

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John doe                | 23  | USA       |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |
| 8 |                         |     |           |

Demostración

### Guardar la información en el servidor

Para hacer más sencillo este ejemplo no voy a guardar la información en una base de datos, pues he de suponer que sabes como hacerlo así que voy a utilizar las “sesiones” para simular que guardamos y actualizamos la información, el código para lograr esto es el siguiente:

```

1. <?php
2. header("Content-Type: text/plain");
3. session_start();
4.
5. $add = $_POST['records'];
6.
7. if(!isset($_SESSION['data'])){
8. $data = array(//creates the initial data
9. 'success'=>true,
10. 'total'=>11,
11. 'data'=>array(
12. array('id'=>1,'name'=>'John doe','age'=>23,'country'=>'USA'),
13. array('id'=>2,'name'=>'Taylor Swift','age'=>19,'country'=>'USA'),
14. array('id'=>3,'name'=>'Carlos Mena','age'=>22,'country'=>'México'),
15. array('id'=>4,'name'=>'Christiano Ronaldo','age'=>24,'country'=>'Portugal'),
16. array('id'=>5,'name'=>'Sasha Cohen','age'=>25,'country'=>'USA'),
17. array('id'=>6,'name'=>'Christian Van Der Henst','age'=>27,'country'=>'Guatemala'),
18. array('id'=>7,'name'=>'Collis Ta'eed','age'=>31,'country'=>'USA')
19.)
20.);
21. $_SESSION['data'] = $data; //load the data in sessions for the first time
22. }else{
23. $data = $_SESSION['data']; //get the data if exist in session
24. }
25.
26. if(isset($add)){ //if there are records to insert/update
27. $records = json_decode(stripslashes($add)); //parse the string to PHP objects
28. $ids = array();
29. foreach($records as $record){
30. if(isset($record->newRecordId)){ //records to insert
31. $id = count($data['data']);
32. $info = array(
33. 'id'=> id,
34. 'name'=> $record->name,
35. 'age'=> $record->age,
36. 'country'=> $record->country
37.);

```

```

38.
39. array_push($data['data'],$info); //add the new record to session
40. array_push($ids,array('oldId'=>$record->newRecordId,'id'=>$id));//new id
41. }else{ //records to update
42. foreach($data['data'] as $key=>$r){ //search the record to update
43. if($r['id'] == $record->id){
44. $data['data'][$key]['name'] = $record->name; //update the properties
45. $data['data'][$key]['age'] = $record->age;
46. $data['data'][$key]['country'] = $record->country;
47. break;
48. }
49. }
50. }
51. }
52.
53. //print the success message
54. echo json_encode(array(
55. 'success'=>true,
56. 'data'=>$ids
57.));
58. }else{
59. //print all records in session
60. echo json_encode($data);
61. }
62. ?>

```

He comentado los puntos más importantes en el código, recuerda que es solo un ejemplo, en el mundo real deberías guardar la información en una base de datos o enviarlas a un WebService, etc. cuando implementes tu “controller” debes tener en cuenta que éste debe hacer dos cosas (por ahora).

- Si recibe registros (mediante el parámetro “post” llamado “records”) debe insertarlos o actualizarlos, esto depende de la variable “newRecordId” ya que si existe nos indica que es un registro nuevo y tiene que insertarse de lo contrario es un registro existente y tiene que actualizarse.
- Si no recibe registros solamente debe desplegar los registros que existen.

### Creación del Grid editable

En este momento vamos a crear el Grid editable con los campos que definimos en el código anterior; escribiremos en el método “init” el siguiente código:

```

1. init: function(){
2. //main code
3. //creates the store for the grid
4. var store = new Ext.data.JsonStore({
5. url: 'editorgrid-ajax.php',
6. root: 'data',
7. id:'id',
8. fields: ['name','age','country']
9. });
10. //load the data
11. store.load();
12.
13. //creates the texfield to edit the data
14. var textField = new Ext.form.TextField();
15. var numberField = new Ext.form.NumberField({allowBlank:false});
16. //creates the editor grid
17. this.grid = new Ext.grid.EditorGridPanel({
18. store: store,
19. columns: [
20. new Ext.grid.RowNumberer(),

```

```

21. {header:'Name', dataIndex:'name', sortable: true, width:145, editor:textField},
22. {header:'Age', dataIndex:'age', sortable: true, editor:numberField},
23. {header:'Country', dataIndex:'country', sortable: true, editor:textField}
24.],
25. border: false,
26. stripeRows: true
27. });
28.
29. //creates a window to hold the grid
30. var win = new Ext.Window({
31. title: 'Editor Grid example',
32. layout: 'fit',
33. width: 410,
34. height:350,
35. items: this.grid
36. });
37. //show the window and the grid
38. win.show();
39. },

```

No me detendré a explicar el código anterior pues ya lo hice en los tutoriales anteriores, así que si no tienes idea de lo que pasó, te recomiendo comenzar con los tutoriales pasados.

Hasta ahora deberíamos ver en pantalla algo como la siguiente imagen:

|  | Name | Age | Country | |
|---|---|---|---|---|
| 1 | John doe | 23 | USA |
| 2 | Taylor Swift | 19 | USA |
| 3 | Carlos Mena | 22 | México |
| 4 | Christiano Ronaldo | 24 | Portugal |
| 5 | Sasha Cohen| | 25 | USA |
| 6 | Christian Van Der Henst | 27 | Guatemala |
| 7 | Collis Ta'eed | 31 | USA |

Grid editable

### Guardar los registros modificados

Ahora si viene la parte interesante de este tutorial, pero primero necesitamos crear un botón para que al ser presionado por el usuario envíemos la información al servidor mediante Ajax, también vamos a crear un "ToolBar" donde estarán los botones que usaremos.

```

1. var win = new Ext.Window({
2. title: 'Editor Grid example',
3. tbar:[
4. defaults:{scope:this},
5. items:[
6. {text:'Save changes',iconCls:'save-icon',handler:this.save}
7.]
8. },
9. layout: 'fit',
10. width: 410,
11. height:350,
12. items: this.grid
13. });

```

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John doe                | 23  | USA       |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |

Toobar en la ventana contenedora

Solamente quiero hacer notar que mediante la propiedad “handler” definimos la función que se ejecutará cuando el usuario de clic sobre el botón, además mediante la propiedad “default” podemos aplicar la propiedad “scope” a todos los botones que definamos, es importante definir el “scope” para la correcta ejecución de la función.

Dentro de la función “save” escribiremos el siguiente código:

```

1. //save changes in the grid
2. var modified = this.grid.getStore().getModifiedRecords(); //step 1
3. if(!Ext.isEmpty(modified)){
4. var recordsToSend = [];
5. Ext.each(modified, function(record) { //step 2
6. recordsToSend.push(Ext.apply({id:record.id},record.data));
7. });
8.
9. this.grid.el.mask('Saving...', 'x-mask-loading'); //step 3
10. this.grid.stopEditing();
11.
12. recordsToSend = Ext.encode(recordsToSend); //step 4
13.
14. Ext.Ajax.request({ // step 5
15. url : 'editorgrid-ajax.php',
16. params :{records : recordsToSend},
17. scope:this,
18. success : function(response) { //step 6
19. this.grid.el.unmask();
20. this.grid.getStore().commitChanges();
21. }
22. });
23. }

```

En el paso 1 se recogen los registros que han sido modificados, esto nos regresa un arreglo con cada “record” que se modificó, luego verificamos que no este vacío.

En el paso 2 iteramos el arreglo de registros modificados y vamos agregándolos a un arreglo que se llama “recordsToSend” en formato JSON, además incluimos el “id” ya que éste no se encuentra con el resto de la información.

En el paso 3 le ponemos una máscara al Grid con el mensaje de “Saving...” y le agregamos la clase CSS “x-mask-loading” la cual muestra una animación que denota que se esta procesando algo. Mediante el método “stopEditing” nos aseguramos que el usuario no este editando ningún otro campo.

En el paso 4 transformamos el “Array” a “String” para ser enviado al servidor en formato JSON.

En el paso 5 realizamos la petición al servidor por medio de Ajax, definimos la “url” que invocaremos así como la información que se enviará en el parámetro “records”, recuerda que cuando enviamos información utilizando la propiedad “params” la petición se realiza por el método “post” por defecto.

En el paso 6 definimos una función que se ejecutará cuando el servidor responda, dentro de esta función quitamos la máscara que cubría el Grid y hacemos un “commit” de los cambios ocurridos en el Grid, al hacer este “commit” los campos marcados como “sucios” (Dirty) dejarán de serlo.

The screenshot shows a grid titled "Editor Grid example". A row is selected and highlighted with a blue border. The first column contains numerical IDs from 1 to 7. The second column is "Name", the third is "Age", and the fourth is "Country". The "Country" column for ID 1 has a text input field containing "Argentina". A "Save changes" button is visible at the top left of the grid area.

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |

Editando el contenido de las celdas

The screenshot shows the same grid editor. The "Country" cell for ID 1 now contains "Argentina" and is no longer highlighted. A progress dialog box labeled "Saving..." with a circular icon is overlaid on the grid area. The "Save changes" button remains at the top left.

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |

Enviando la información al servidor para ser guardada

The screenshot shows the grid after the save operation. The "Country" cell for ID 1 now contains "Argentina" and is highlighted. The progress dialog is gone, and the "Save changes" button is still present at the top left.

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |

La información ha sido guardada correctamente

## Crear un registro nuevo

En la sección anterior logramos guardar los registros que fueron modificados por el usuario, ahora vamos a insertar nuevos registros en el Grid, para esto necesitamos crear un botón en la barra de herramientas que nos permita realizar lo antes mencionado.

```
1. var win = new Ext.Window({
2. title: 'Editor Grid example',
3. tbar:{
4. defaults:{scope:this},
5. items:[
6. {text:'Save changes',iconCls:'save-icon',handler:this.save},
7. {text:'Add Person',iconCls:'add-icon',handler:this.add} // add button
8.]
9. },
10. layout: 'fit',
11. width: 410,
12. height:350,
13. items: this.grid
14.});
```

Ahora necesitamos implementar el método “add” que está invocando el botón cuando es presionado.

```
1. //add a new row to the grid
2. var position = this.grid.getStore().getCount();
3. var id = Ext.id();
4. var defaultData = { //step 1
5. newRecordId: id
6. };
7. var Person = this.grid.getStore().recordType; //step 2
8. var person = new Person(defaultData,id);
9. this.grid.stopEditing(); //step 3
10. this.grid.getStore().insert(position, person); // step 4
11. this.grid.startEditing(position, 1); //step 5
```

El código anterior debe ser escrito dentro del método “add”.

En el paso 1 se crea un objeto con la información por defecto, en este caso únicamente se está definiendo la propiedad “newRecordId” la cual es utilizada por el servidor para saber si el registro lo debe insertar o actualizar en la base de datos, el valor de esta propiedad ha sido generado mediante “Ext.id()” para tener un id único.

El paso 2 es muy importante y es necesario porque cuando creamos el Store del grid nosotros no definimos ni el “Reader”, ni el “Record” directamente, esto lo hizo “Ext” automáticamente, por lo tanto no tenemos una referencia al “Record” generado, gracias a la propiedad “recordType” del “store” accedemos al “Record” que fue creado por el Framework dinámicamente, una vez que tenemos la referencia al componente “Record” creamos una instancia de éste con la información que generamos en el paso 1 y le asignamos su “id”.

En el paso 3 solamente detenemos cualquier edición activa por parte del usuario.

El paso 4 es el que hace la “magia”, ya que es aquí donde se inserta el nuevo “record” al store del Grid y automáticamente se reflejará la nueva fila vacía en el Grid, el primer parámetro que recibe este método es la posición donde queremos que se inserte, la posición la hemos calculado previamente contando el total de filas existentes esto nos da como resultado que la fila se inserte hasta el final del grid.

En el paso 5 solamente hacemos que de la fila que acabamos de insertar en la columna número “1” aparezca una caja de texto (que definimos al crear el Grid) para editar el contenido de la celda, es importante mencionar que la columna “0” para este caso es la columna donde están los números, en caso de no tener esta columna podríamos comenzar a editar la columna “0”.

Al actualizar nuestro explorador debería ver algo como la siguiente imagen en la pantalla:

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |
| 8 |                         |     |           |

Agregando una fila al Grid

Para guardar la información que insertemos en el registro nuevo no tenemos que hacer nada puesto que ya se debería guardar utilizando el método “save” que hicimos anteriormente.

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |
| 8 | Crysfel Villa           | 25  | México    |

Guardando el registro nuevo

Si has notado se esta enviando al servidor el registro a insertar con la propiedad “newRecordId”, además el “id” asignado no es el mismo “id” que le asigna el servidor, esto lo sabemos porque en la respuesta el servidor nos dice el “id” que recibió y el “id” que asignó.

```

records [{"id": "ext-gen46", "newRecordId": "ext-gen46", "name": "Crysfel Villa", "age": 25}

```

```

{"success":true, "data": [{"oldId": "ext-gen46", "id": 7}]}

```

Parámetros enviado al servidor por Ajax

Lo que debemos hacer ahora es modificar el método “save” para que sincronice los “id’s” correctamente. Dentro de la función “success” escribiremos el siguiente código:

1. //update the records with the correct ID's
2. var info = Ext.decode(response.responseText); // step 1
3. Ext.each(info.data,function(obj){
4.     var record = this.grid.getStore().getById(obj.oldId); //step 2
- 5.
6.     record.set('id',obj.id); //step 3
7.     delete record.data.newRecordId; //step 4
8. },this);

En el paso 1 se está decodificando la información que nos regresó el servidor en formato JSON para poder manipularla fácilmente.

En el paso 2 se obtiene el “Record” utilizando el “id” que le fue asignado a la hora de crearlo.

En el paso 3 se actualiza su “id” por el “id” asignado por el servidor en nuestra base de datos.

El paso 4 es importante para eliminar la propiedad “newRecordId” que tiene el registro, de esta forma evitamos que se inserte nuevamente si solamente queremos actualizarlo.

### **Cancelar los cambios**

Por defecto al editar el contenido de las celdas de un Grid éstas quedan en un estado de “Dirty”, en este estado nosotros podemos deshacer los cambios y regresar a la información original.

Primero creamos el botón que nos permitirá realizar la acción pertinente.

1. var win = new Ext.Window({
2.     title: 'Editor Grid example',
3.     tbar:{
4.         defaults:{scope:this},
5.         items:[
6.             {text:'Save changes',iconCls:'save-icon',handler:this.save},
7.             {text:'Add Person',iconCls:'add-icon',handler:this.add},
8.             {text:'Cancel changes',iconCls:'cancel-icon',handler:this.cancel} //cancel changes

```

9.]
10. },
11. layout: 'fit',
12. width: 410,
13. height:350,
14. items: this.grid
15.);

```

**Editor Grid example**

|   | Name                    | Age | Country   |
|---|-------------------------|-----|-----------|
| 1 | John Doe                | 30  | Argentina |
| 2 | Taylor Swift            | 19  | USA       |
| 3 | Carlos Mena             | 22  | México    |
| 4 | Christiano Ronaldo      | 24  | Portugal  |
| 5 | Sasha Cohen             | 25  | USA       |
| 6 | Christian Van Der Henst | 27  | Guatemala |
| 7 | Collis Ta'eed           | 31  | USA       |
| 8 | Crysfel Villa           | 25  | México    |

Botón para cancelar los cambios

Luego implementamos el método “cancel” de la siguiente manera:

```

1. cancel: function(){
2. //cancel the changes in the grid
3. this.grid.getStore().rejectChanges();
4. }

```

Cuando invocamos el método “rejectChanges” del store, automáticamente regresan al estado original las celdas modificadas, esto es suficiente para deshacer los cambios ocurridos.

### Conclusiones

En este tutorial aprendimos como guardar información a través de Ajax, también como insertar y actualizar los registros de un store y verlos reflejados en un Grid editable, en futuros tutoriales mostraré como hacerlo de una manera más sencilla utilizando las nuevas características de Ext 3.

## Agrupado y Sumatoria de Columnas

En este tutorial se hablará de cómo podemos agrupar en un Grid la información que cuenta con un atributo en común. Veremos cómo podemos sumar los campos que representan cierta cantidad de dinero o artículos y mostrar ese resultado en una fila de nuestro Grid.

La siguiente imagen es un ejemplo de lo que tendremos al final de este tutorial:

| Destinations Summary     |          |            |          |           |            |
|--------------------------|----------|------------|----------|-----------|------------|
| Country                  | Hotel    | Plane      | Food     | Souvenirs | Total Cost |
| <b>Continent: Ameria</b> |          |            |          |           |            |
| USA                      | \$300.00 | \$900.00   | \$150.00 | \$200.00  | \$1,550.00 |
| Mexico                   | \$200.00 | \$500.00   | \$150.00 | \$100.00  | \$950.00   |
| (2 Countries)            | \$500.00 | \$1,400.00 | \$300.00 | \$300.00  | \$2,500.00 |
| <b>Continent: Europe</b> |          |            |          |           |            |
| Spain                    | \$300.00 | \$1,100.00 | \$200.00 | \$100.00  | \$1,700.00 |
| (1 Country)              | \$300.00 | \$1,100.00 | \$200.00 | \$100.00  | \$1,700.00 |

Resultado Final

Durante el tutorial se irá explicando el código necesario para realizar lo que se muestra en la imagen, recuerda que puedes descargar el código fuente si es necesario.

### La base de datos

La información para el Grid estará en una base de datos de MySQL que contendrá una tabla llamada "destinations", en esta tabla solo tenemos la información para poder trabajar a lo largo de este tutorial. El código para generar la tabla se muestra a continuación:

```
1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Servidor: localhost
6. -- Tiempo de generación: 10-11-2010 a las 06:51:06
7. -- Versión del servidor: 5.1.36
8. -- Versión de PHP: 5.3.0
9.
10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11.
12. --
13. -- Base de datos: `test`
14. --
15. -----
16. --
17. -- Estructura de tabla para la tabla `destinations`
18. --
19.
20. CREATE TABLE IF NOT EXISTS `destinations` (
21. `id` int(11) NOT NULL AUTO_INCREMENT,
22. `continent` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
23. `country` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
24. `hotel` int(11) NOT NULL,
```

```

25. `plane` int(11) NOT NULL,
26. `food` int(11) NOT NULL,
27. `souvenirs` int(11) NOT NULL,
28. `cost` int(11) NOT NULL,
29. PRIMARY KEY (`id`)
30.) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT
 =10 ;
31.
32. --
33. -- Volcar la base de datos para la tabla `destinations`
34. --
35.
36. INSERT INTO `destinations` (`id`, `continent`, `country`, `hotel`, `plane`, `food`, `souvenirs`, `cost`)
) VALUES
37. (1, 'Ameria', 'USA', 300, 900, 150, 200, 0),
38. (2, 'Europe', 'Spain', 300, 1100, 200, 100, 0),
39. (3, 'Ameria', 'Mexico', 200, 500, 150, 100, 0);

```

La base de datos se llama “test” pero puedes usar el nombre que te agrade, solo recuerda cambiar el nombre cuando se haga la conexión mediante PHP.

### **Exponer la información**

Como la información está almacenada en una base de datos, necesitamos realizar la conexión vía PHP o cualquier otro lenguaje de servidor y un simple “SELECT”. El siguiente código se debe escribir en un archivo llamado “serverside/getGroups.php”.

```

1. <?php
2. $connection= mysql_connect("localhost","root","");
3. or die("Connection Failed".mysql_error());
3. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
4.
5. $result= mysql_query("SELECT * FROM destinations"); //step 1
6.
7. $data= array();
8.
9. while($row= mysql_fetch_array($result)){ //step 2
10. array_push($data,array(
11. "id" => $row["id"],
12. "continent" => $row["continent"],
13. "country" => $row["country"],
14. "hotel" => $row["hotel"],
15. "plane" => $row["plane"],
16. "food" => $row["food"],
17. "souvenirs"=> $row["souvenirs"],
18. "cost" => $row["cost"]
19.));
20. }
21.
22. //step 3
23. echo json_encode(
24. array(
25. "success" => true,
26. "data" => $data
27.));

```

En el paso uno se hace el query que regresa todo lo que contiene la tabla “destinations”, es un query muy sencillo.

En el paso dos se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo con la información de cada uno de los contactos.

En el paso cuatro se imprime la información en formato JSON, la respuesta será como el siguiente ejemplo:

```
1. {"success":true,"data":[{"id":"1","continent":"Ameria","country":"USA","hotel":300,"plane":900,"food":150,"souvenirs":200,"cost":0},{"id":2,"continent":"Europe","country":"Spain","hotel":300,"plane":1100,"food":200,"souvenirs":100,"cost":0},{"id":3,"continent":"Ameria","country":"Mexico","hotel":200,"plane":900,"food":150,"souvenirs":100,"cost":0}]]
```

Esto es suficiente para poder generar la interface del Grid mediante JavaScript ya que tenemos toda la información necesaria.

### Preparando el Entorno

Para poder usar el GridSummary es necesario importar algunos archivos en nuestro archivo HTML (groupSummary-example.html). Los archivos que necesitamos son los siguientes: GroupSummary.css y GroupSummary.js.

```
1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3.
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
7. <title>Demo: Group Summary Example | Quizzpot</title>
8.
9. <link rel="stylesheet" type="text/css" href="../ext-3.3.0/resources/css/ext-all.css" />
10. <link rel="stylesheet" type="text/css" href="../ext-3.3.0/examples/ux/css/GroupSummary.css" />
11.
12. <script type="text/javascript" src="../ext-3.3.0/adapter/ext/ext-base.js"></script>
13. <script type="text/javascript" src="../ext-3.3.0/ext-all.js"> </script>
14. <script type="text/javascript" src="../ext-3.3.0/examples/ux/GroupSummary.js"></script>
15.
16. <script type="text/javascript" src="Group-Summary.js"></script>
17.
18. </head>
19. <body>
20. </body>
21. </html>
```

Como podemos notar estos archivos los encontramos en el directorio donde está la librería de Ext JS en la carpeta de ejemplos donde encontraremos la carpeta ux. Si notamos solo es un archivo de estilos (.css) y un archivo de JavaScript, con estos archivos podremos cumplir lo propuesto en este tutorial.

También podríamos copiarlos en una carpeta “plugins” que tengamos en nuestro proyecto, de esta manera cuando se realice el deploy del proyecto eliminamos los ejemplos, para este ejemplo los dejaremos en el directorio “ejemplos” solo para mostrar el lugar exacto donde se encuentran.

### Empaquetando el Tutorial

Ya en nuestro archivo de JavaScript (Group-Summary) vamos a empaquetar el código para evitar conflictos con otras variables o librerías. Esto es algo muy importante y siempre lo debemos realizar.

```
1. Ext.ns("com.quizzpot.tutorials");
2.
3. com.quizzpot.tutorials.Group = {
4. init : function() {
5. //code
6. }
7. }
```

```
8. Ext.onReady(com.quizzpot.tutorials.Group .init,com.quizzpot.tutorials.Group);
```

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### Preparando la Información para el GRID

Lo primero que haremos es preparar la información que recibimos de la base de datos. Lo que necesitamos hacer es crear un JsonReader para poder interpretar la información que recibimos del servidor:

```
1. var reader = new Ext.data.JsonReader({ //step 1
2. totalProperty : 'total',
3. successProperty : 'success',
4. messageProperty : 'message',
5. idProperty : 'id',
6. root : 'data'
7. },
8. //step 2
9. {name: "continent", type: "string"},
10. {name: "country", type: "string"},
11. {name: "hotel", type: "float"},
12. {name: "plane", type: "float"},
13. {name: "food", type: "float"},
14. {name: "souvenirs", type: "float"},
15. {name: "cost", type: "float"}
16.]
17.);
```

En el paso uno lo que hacemos es crear una instancia del “JsonReader” este componente nos permitirá leer e interpretar la respuesta del servidor para cada petición Ajax realizada. Lo que tenemos que notar es el atributo “messageProperty” que define la propiedad que aloja un mensaje en la respuesta del servidor, otra propiedad importante es “successProperty” la cual indicará si la operación ha sido realizada con éxito.

En el paso dos colocamos los campos que conforman el “Record” que contendrá la información que usaremos en el Grid.

Lo siguiente es hacer el store que tendrá al “JsonReader”, en esta ocasión veremos un nuevo tipo de store, es el “GroupingStore” este tipo de store es utilizado para agrupar la información de un Grid. Su implementación es la siguiente:

```
1. this.gstore =new Ext.data.GroupingStore({ //step 1
2. url : "serverside/getGroups.php", //step 2
3. reader : reader,
4. sortInfo : {field:"continent", direction:"ASC"},
5. groupField : "continent"
6. });
7.
8. this.gstore.load();
```

En el paso uno lo que hacemos es crear la instancia del “GroupingStore”, lo que este store nos permite hacer es agrupar los registros usando uno de los campos disponibles. Notemos el atributo “groupField”, este es el atributo que se encarga de hacer lo antes mencionado.

En el paso dos configuramos el “GroupingStore” de la misma manera como se configura un “JsonStore” o un “Store” a excepción del atributo “groupField” del cual hicimos mención hace uno momentos. Y por último cargamos el store.

### Creando el GRID

Ya que tenemos todo listo, lo que debemos hacer es configurar el Grid para mostrar la información agrupada. Lo haremos de la siguiente manera:

```
1. //step 1
2. var textField = new Ext.form.TextField({allowBlank: false,}),
3. numField = new Ext.form.NumberField({allowBlank: false, allowNegative: false});
4.
5. this.grid = new Ext.grid.EditorGridPanel({ //step 2
6. store : this.gstore, // le pasamos el GroupStore
7. columns : [//configuración de las columnas del Grid
8. {header : "Continent",
9. dataIndex : "continent",
10. hideable : false,
11. groupable : true,
12. width : 100,
13. editor : textField,
14. },
15. {header : "Country",
16. dataIndex : "country",
17. groupable : true,
18. editor : textField,
19. },
20. {header : "Hotel",
21. dataIndex : "hotel",
22. width : 100,
23. sortable : true,
24. groupable : false,
25. renderer : Ext.util.Format.usMoney,
26. editor : numField,
27. },
28. {header : "Plane",
29. dataIndex : "plane",
30. width : 100,
31. sortable : true,
32. groupable : false,
33. renderer : Ext.util.Format.usMoney,
34. editor : numField
35. },
36. {header : "Food",
37. dataIndex : "food",
38. width : 100,
39. sortable : true,
40. groupable : false,
41. renderer : Ext.util.Format.usMoney,
42. editor : numField,
43. },
44. {header : "Souvenirs",
45. dataIndex : "souvenirs",
46. width : 100,
47. sortable : true,
48. groupable : false,
49. renderer : Ext.util.Format.usMoney,
50. editor : numField,
51. },
52. {header : "Total Cost",
53. dataIndex : "cost",
54. width : 100,
55. sortable : true,
56. groupable : false
57. }
58. }
```

```

58.], //step 3
59. view : new Ext.grid.GroupingView({
60. forceFit : true,
61. ShowGroupName : true,
62. enableNoGroup : false,
63. enableGropingMenu : false,
64. hideGroupedColumn : true
65. }),
66. });

```

En el paso uno creamos los campos “textField” y “numField” esto con el fin de ponerlo en las columnas del Grid para que este pueda ser editado.

En el paso dos solo hacemos una instancia del EditorGridPanel, y realizamos la configuración necesaria para el Grid, en el atributo store le pasamos el nuevo store como lo haríamos normalmente con los otros tipos de store.

No entraremos en detalle de la configuración de las columnas del Grid pero si haremos mención de los dos nuevos atributos que usamos en la configuración de las columnas, estos son: groupable y renderer.

El primero es un tributo te tipo Boolean, éste nos permite decidir si deseamos que esta columna pueda ser usada para agrupar la información del Grid.

La propiedad renderer nos permite darle un formato a los campos de la columna como en esta ocasión estamos trabajando con números que representan dinero usamos “Ext.util.Format.usMoney”, si quieres conocer más al respecto de esta propiedad te recomiendo leer el tutorial que ya hemos publicado al respecto.

En el paso tres hacemos el “view” que es usado por el Grid para algunas configuraciones extra relacionadas con la manera de como agrupara la información.

Con esto ya tenemos nuestro Grid, los siguiente mostrarlo en una ventana:

```

1. var win = new Ext.Window({
2. title : "Destinations Summary",
3. layout : "fit",
4. width : 550,
5. height : 300,
6. items : this.grid
7. });
8.
9. win.show();

```

Con esto tendríamos algo como lo que se muestra en la siguiente imagen:

| Country                  | Hotel    | Plane      | Food     | Souvenirs | Total Cost |
|--------------------------|----------|------------|----------|-----------|------------|
| <b>Continent: Ameria</b> |          |            |          |           |            |
| USA                      | \$300.00 | \$900.00   | \$150.00 | \$200.00  | 0          |
| Mexico                   | \$200.00 | \$500.00   | \$150.00 | \$100.00  | 0          |
| <b>Continent: Europe</b> |          |            |          |           |            |
| Spain                    | \$300.00 | \$1,100.00 | \$200.00 | \$100.00  | 0          |

Grid Agrupado

Con esto terminas el primer objetivo de nuestro tutorial. Como podemos ver los registros están agrupados bajo el campo “continente” como fue especificado en el GroupingStore.

### Group Summary

Lo siguiente es crear el resumen de la información del Grid. Para esto usaremos el plugin Summary como se muestra a continuación.

```
1. var textField = new Ext.form.TextField({allowBlank: false,}),
2. numField = new Ext.form.NumberField({allowBlank: false, allowNegative: false});
3.
4. //step 1
5. var summary = new Ext.ux.grid.GroupSummary();
6.
7. //step 2
8. Ext.ux.grid.GroupSummary.Calculations["totalCost"] =function(v, record, field){
9. return v + (record.data.hotel + record.data.plane + record.data.food + record.data.souvenirs);
10. };
11.
12. this.grid = new Ext.grid.EditorGridPanel({
13. store : this.gstore,
14. plugins : summary, //step 3
15. columns : [
16. //.... Configuración del las columnas del grid.
17.]
18. });
```

El código anterior está un poco resumido, pero son los mismos pasos que la sección anterior, solo le agregaremos lo que se describe a continuación.

En el paso uno lo que hacemos es crear una instancia del GroupSummary para poder mostrar los resúmenes de las columnas del Grid.

En el paso dos lo que hacemos es crear una nueva forma de “calculo”, como podemos ver le estamos dando el nombre de “totalCost” lo único que esto hace es ir a la columna, buscar en la fila que se especifica, tomar su valor y sumarlo a la siguiente. Es importante mencionar que es posible hacer distintos tipos de “cálculos” de esta manera.

En el paso tres agregamos el Summary via plugin, con esto ya podremos usar los “cálculos” disponibles para los distintos tipos de información que se pueden manejar.

Lo que sigue es realizar estos cálculos, esto ya se hace directamente en las columnas.

```
1. this.grid = new Ext.grid.EditorGridPanel({
2. store : this.gstore,
3. plugins : summary,
4. columns : [
5. {header : "Continent",
6. dataIndex : "continent",
7. hideable : false,
8. groupable : true,
9. width : 100,
10. editor : textField,
11. },
12. {header : "Country",
13. dataIndex : "country",
14. groupable : true,
15. editor : textField,
16. //step 1
17. summaryType : "count",
18. summaryRenderer: function(v, params, data){
```

```

19. return ((v === 0 || v > 1) ? '(' + v +' Countries)' : '(1 Country)');
20. }
21. },
22. header : "Hotel",
23. dataIndex : "hotel",
24. width : 100,
25. sortable : true,
26. groupable : false,
27. editor : numField,
28. renderer : Ext.util.Format.usMoney,
29. //step 2
30. summaryType : "sum",
31. render : function(v){
32. return v;
33. }
34. },
35. header : "Plane",
36. dataIndex : "plane",
37. width : 100,
38. sortable : true,
39. groupable : false,
40. editor : numField,
41. renderer : Ext.util.Format.usMoney,
42. summaryType : "sum",
43. render : function(v){
44. return v;
45. }
46. },
47. header : "Food",
48. dataIndex : "food",
49. width : 100,
50. sortable : true,
51. groupable : false,
52. editor : numField,
53. renderer : Ext.util.Format.usMoney,
54. summaryType : "sum",
55. render : function(v){
56. return v;
57. }
58. },
59. header : "Souvenirs",
60. dataIndex : "souvenirs",
61. width : 100,
62. sortable : true,
63. groupable : false,
64. editor : numField,
65. renderer : Ext.util.Format.usMoney,
66. summaryType : "sum",
67. render : function(v){
68. return v;
69. }
70. },
71. header : "Total Cost",
72. dataIndex : "cost",
73. width : 100,
74. sortable : true,
75. groupable : false,
76. //step 3
77. renderer: function(v, params, record){
```

```

78. return Ext.util.Format.usMoney(record.data.hotel + record.data.plane + record.data.food
+ record.data.souvenirs);
79. },
80. //step 4
81. summaryType : "totalCost",
82. summaryRenderer: Ext.util.Format.usMoney
83. }
84.],
85. //..el código del view esta aquí
86. });

```

En el paso uno usamos “summaryType” para especificar qué tipo de cálculo queremos hacer, para este registro usaremos “count” esto con el motivo de saber cuántos registros tenemos en esta columna.

El atributo “summaryRenderer” se encarga de mostrar el resultado del calculo que se realizo en el “summaryType”.

En el paso dos usamos “summaryType” con “sum”, con esto sumamos todos los registros que se encuentran en esta columna. La única diferencia con el paso anterior es que usamos “render” para imprimir el resultado en el Grid, esto es porque no le estamos concatenando nada a la respuesta.

Las demás columnas tienen los mismos atributos que en el paso dos ya que también queremos conocer la suma de esos campos.

En el paso tres pasa algo interesante, lo primero que hacemos es hacer la suma de todos los campos que pertenecen al registro que se encuentra en esa fila, desplegamos el resultado en la columna “Total Cost” de ese registro. Para esto usamos el atributo “renderer” ya que estamos haciendo una sumatoria de los campos sin el uso de un “summaryType”.

En segundo lugar usamos el “summaryType” que creamos hace unos momentos atrás, y usamos un “summaryRenderer” porque queremos modificar el resultado del “summaryType”, lo único que le agregamos es el signo de dinero.

Con esto terminamos el segundo objetivo de este tutorial, hasta este momento se tiene que tener algo semejante a esto:

| Destinations Summary     |          |            |          |           |            |
|--------------------------|----------|------------|----------|-----------|------------|
| Country                  | Hotel    | Plane      | Food     | Souvenirs | Total Cost |
| <b>Continent: Ameria</b> |          |            |          |           |            |
| USA                      | \$300.00 | \$900.00   | \$150.00 | \$200.00  | \$1,550.00 |
| Mexico                   | \$200.00 | \$500.00   | \$150.00 | \$100.00  | \$950.00   |
| (2 Countries)            | \$500.00 | \$1,400.00 | \$300.00 | \$300.00  | \$2,500.00 |
| <b>Continent: Europe</b> |          |            |          |           |            |
| Spain                    | \$300.00 | \$1,100.00 | \$200.00 | \$100.00  | \$1,700.00 |
| (1 Country)              | \$300.00 | \$1,100.00 | \$200.00 | \$100.00  | \$1,700.00 |

Summaries

Notemos algo interesante que sucede cuando editamos una de las cantidades, los campos de la columna “Total Cost” son actualizados automáticamente esto gracias a lo que sucede en la primera parte del paso tres que mencionamos hace unos instantes.

### Guardando la Información

Lo que debemos hacer ahora es guardar los registros conforme van siendo editados ya que si alguno de estos registros es editado no se mostrará la información la siguiente vez que se abra la aplicación. Para esto usaremos nuestra base de datos, lo primero que haremos es agregar un “listener” al Grid que nos dirá los campos que han sido modificados y por último modificaremos esos registros en la base de datos.

Usaremos el evento “afteredit” que se disparará después de que se ha editado una celda.

```
1. this.grid = new Ext.grid.EditorGridPanel({
2. //configuración previo del grid
3. });
4.
5. this.grid.on('afteredit', this.afterEdit, this);
```

Como podemos notar lo único que hacemos es decirle que cada vez que se edite algo en la tabla se llame a la función “afterEdit” esto mediante el evento “afteredit” también es importante mencionar que mediante “this” indicamos el scope en el que se ejecutará la función “afterEdit”. Esta función se encarga de obtener los nuevos valores de los registros editados.

```
1. afterEdit : function(e){
2. //step 1
3. var recordsToSend = [];
4. recordsToSend = Ext.encode(e.record.data);
5.
6. this.grid.el.mask("Saving...","x-mask-loading");
7. Ext.Ajax.request({
8. scope : this,
9. url : "serverside/saveGroups.php",
10. params : {records:recordsToSend},
11. success : this.onSuccess
12. });
13. },
```

Se crea la función que recibe como parámetro lo que nos manda el evento “afteredit”. En la función se crea el arreglo “recordsToSend”, con el fin de guardar en este arreglo los valores modificados.

En el paso uno se realiza lo antes mencionado, usando los parámetros que mandó el evento, obtenemos la información y la codificamos para poder mandarla en la petición Ajax.

En el paso dos se hace la petición Ajax, se especifica mediante el atributo “url” el archivo que tratará esta consulta. Y con “params” le decimos que es lo que enviará a ese archivo, es necesario también mencionar que en “scope” se indica donde se ejecutará la función indicada en “success”.

Esta función tiene la siguiente estructura:

```
1. onSuccess : function(response,options){
2. this.grid.el.unmask();
3. this.grid.getStore().commitChanges();
4. }
```

Esta función se encarga de quitar la máscara que colocamos en el Grid mientras se ejecuta la petición Ajax. También se encarga de mostrar los cambios que sufrió el Grid.

El código del archivo responsable de guardar la información en la base de datos se muestra a continuación:

```
1. <?php
2. $connection=mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
4.
5. $info = $_POST["records"];
6. //step 1
7. $data = json_decode(stripslashes($info));
8.
9. //srtep 2
```

```

10. $continent = $data->continent;
11. $country = $data->country;
12. $hotel = $data->hotel;
13. $plane = $data->plane;
14. $food = $data->food;
15. $souvenirs = $data->souvenirs;
16. $cost = $data->cost;
17.
18. //step 3
19. $query = sprintf("UPDATE destinations SET continent = '%s', country = '%s', hotel = %d, plane
= %d, food = %d, souvenirs = %d, cost = %d WHERE country=%s",
20. mysql_real_escape_string($continent),
21. mysql_real_escape_string($country),
22. mysql_real_escape_string($hotel),
23. mysql_real_escape_string($plane),
24. mysql_real_escape_string($food),
25. mysql_real_escape_string($souvenirs),
26. mysql_real_escape_string($cost),
27. mysql_real_escape_string($country));
28.
29. $rs = mysql_query($query);
30.
31. //step 4
32. echo json_encode(array(
33. "success" => mysql_errno() == 0,
34. "msg" => mysql_errno() == 0?"redor updated successfully":mysql_error()
35.));

```

En el paso uno, una vez que recibimos la información que mandó la petición Ajax la decodificamos con la función `json_decode()` de PHP, con esto ya podemos acceder a la información que fue editada en el “Grid”. La función “stripslashes” elimina el carácter “\” que Apache le agrega, es posible que la configuración de tu servidor lo haga, pero es bueno prevenir esto para evitar errores en el futuro.

En el paso tres obtenemos la información y la separamos en variables para poder insertarlas a la base de datos, esto es opcional ya que podríamos utilizar el mismo objeto `$data`.

En el paso cuatro hacemos un “update” a la tabla “destinations” que es donde esta grabada la información que desplegamos en el Grid y le damos como valores la información que fue mandada mediante Ajax.

En el paso cinco lo que hacemos es mandar un mensaje de éxito si es que todo sale bien, de lo contrario se manda el mensaje de error de MySQL.

Con esto los registros editados serán guardados en la base de datos.

## Conclusión

Hoy vimos como podemos agrupar los registro de un Grid que tienen un campo en común, también vimos como podemos usar el Summary plugin para poder resumir un la información del Grid. Esto es algo que le dará más vista a nuestras aplicaciones, una mejor experiencia a los usuarios.

## CRUD de un catálogo de contactos

En esta ocasión vamos a Crear, Leer, Actualizar y Borrar (CRUD) un catálogo de contactos usando un EditorGrid que a su vez contendrá un Writer en su Store. Con esto podremos simplificar nuestro código al momento de hacer las peticiones a nuestro servidor.

En este tutorial haremos un CRUD para poder editar la información contenida en un EditorGrid, esto lo lograremos mediante el Writer de un store.

The screenshot shows a window titled 'CRUD Example'. At the top, there are two buttons: a green '+' icon labeled 'New Contact' and a red '-' icon labeled 'Delete'. Below the buttons is a table with three columns: 'E-mail', 'Name', and 'Lastname'. The table contains four rows of data:

| E-mail               | Name      | Lastname |
|----------------------|-----------|----------|
| alejandro@gmail.com  | Alejandro | Cruz     |
| jose@hotmail.com     | Jose      | Paz      |
| pedro_2010@gmail.com | Pedro     | Lopez    |

Resultado Final

Durante el tutorial se irá explicando el código necesario para realizarlo, recuerda que puedes descargar el código fuente si te es necesario.

### La Base de Datos

La información para el EditorGrid estará en una base de datos de MySQL que contendrá una tabla llamada "contacts", en esta tabla solo tenemos la información básica de un contacto. El código para generar la tabla se muestra a continuación.

```
1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Servidor: localhost
6. -- Tiempo de generación: 18-10-2010 a las 13:25:06
7. -- Versión del servidor: 5.1.36
8. -- Versión de PHP: 5.3.0
9.
10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11. --
12. -- Base de datos: `test`
13. --
14. -----
15. --
16. -- Estructura de tabla para la tabla `contacts`
17. --
18.
19. CREATE TABLE IF NOT EXISTS `contacts` (
20. `id` int(11) NOT NULL AUTO_INCREMENT,
21. `email` varchar(20) COLLATE utf8_unicode_ci NOT NULL,
22. `firstName` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
23. `lastName` varchar(39) COLLATE utf8_unicode_ci NOT NULL,
24. PRIMARY KEY (`id`)
25.) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT
 =10 ;
26.
27. --
```

```

28. -- Volcar la base de datos para la tabla `contacts`
29. --
30.
31. INSERT INTO `contacts` (`id`, `email`, `firstName`, `lastName`) VALUES
32. (1, 'pedro@gmail.com<script type="text/javascript">
33. /* <![CDATA[*/
34. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})());
35. /*]]> */
36. </script>', 'Pedro', 'Lopez'),
37. (2, 'jose@hotmail.com<script type="text/javascript">
38. /* <![CDATA[*/
39. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})());
40. /*]]> */
41. </script>', 'Jose', 'Paz'),
42. (3, 'john@hotmail.com<script type="text/javascript">
43. /* <![CDATA[*/
44. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l);}}catch(e){}})());
45. /*]]> */
46. </script>', 'John', 'Smith');

```

La base de datos se llama “test” pero puedes usar el nombre que te agrade, solo recuerda cambiar el nombre cuando se haga la conexión mediante PHP.

### **Exponer la Información**

Como la información está almacenada en una base de datos, necesitamos realizar la conexión vía PHP o cualquier otro lenguaje de servidor y un simple “SELECT” de la siguiente manera. El siguiente código se debe escribir en un archivo llamado “serverside/getContacts.php”.

```

1. <?php
2. //step 1
3. $connection= mysql_connect("localhost","root","");
4. or die("Connection Failed".mysql_error());
5. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
6.
7. $result= mysql_query("SELECT * FROM contacts"); //step 2
8.
9. $data= array();
10. while($row= mysql_fetch_array($result)){ //step 3
11. array_push($data,array(
12. "id" => $row["id"],
13. "first" => $row["firstName"],
14. "last" => $row["lastName"],
15. "email" => $row["email"],
16.));
17. }
18. echo json_encode(//step 4
19. array(
20. "success" => true,
21. "data" => $data
22.));

```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos se hace el query que regresa todo lo que contiene la tabla “contacts”, es un query muy sencillo.

En el paso tres se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo con la información de cada uno de los contactos.

En el paso cuatro se imprime la información en formato JSON, la respuesta será como el siguiente ejemplo:

```
1. {"success":true,"data":[{"id":"1","first":"Pedro","last":"Lopez","email":"pedro@gmail.com<script type="text/javascript">
2. /* <![CDATA[*/
3. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}catch(e){}})();
4. /*]]> */
5. </script>},{"id":"2","first":"Jose","last":"Paz","email":"jose@hotmail.com<script type="text/javascript">
6. /* <![CDATA[*/
7. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}catch(e){}})();
8. /*]]> */
9. </script>"}}
```

Con eso es suficiente para poder generar mediante JavaScript la interface del Grid ya que tenemos toda la información necesaria.

### Empaquetando el Tutorial

Vamos a empaquetar el código para evitar conflictos con otras variables. Esto es algo muy importante y siempre lo debemos realizar.

```
1. Ext.ns("com.quizzpot.tutorials");
2.
3. com.quizzpot.tutorials.Crud = {
4. init : function() {
5. //code
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorials.Crud.init,com.quizzpot.tutorials.Crud);
```

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### Creado el CRUD

Lo que haremos a continuación es solicitar la información al servidor, esto será ya dentro de la función “init” de nuestro archivo “crudexample.js”. Como mencionamos anteriormente usaremos un writer que se encargue de hacer las operaciones básicas (leer, escribir, modificar y eliminar). Para lograr esto es necesario definir la propiedad “api” con las URLs donde el “proxy” realizará las peticiones Ajax de manera automática, nótese que no estamos usando la propiedad “url” que normalmente usamos.

```
1. var proxy = new Ext.data.HttpProxy({ //step 1
2.
3. api: { //step 2
4. read : "serverside/getContacts.php",
5. create : "serverside/createContact.php",
```

```

6. update : "serverside/updateContact.php",
7. destroy : "serverside/destroyContact.php"
8. }
9. });

```

En el paso uno se crea una instancia de “Ext.data.HttpProxy” esto es para que el “store” realice las peticiones Ajax de manera automática.

En el paso dos usamos la configuración “api” con las URLs a donde se realizarán las peticiones Ajax para cada acción a realizar. Por el momento solo tenemos definido el archivo “getContacts.php”, los demás los definiremos más adelante.

A continuación haremos el “Reader” y el “Writer”:

```

1. var reader = new Ext.data.JsonReader({ //step 1
2. totalProperty : "total",
3. successProperty : "success", // indica la propiedad que define si se ha insertado/actualizado o
4. borrado con éxito
5. messageProperty : "message",
6. idProperty : "id",
7. root : "data" //este es el nombre del parámetro que llega al servidor con el JSON modific
8. ado
9. },[//step 2
10. {name: "email", allowBlank: false},
11. {name: "first", allowBlank: false},
12. {name: "last", allowBlank: false}
13.]);
14.
15. var writer = new Ext.data.JsonWriter({ //step 3
16. encode : true,
17. writeAllFields : true//decide si se manda al servidor solamente los campos modificados o tod
o
18. });
19.
20. });

```

En el paso uno lo que hacemos es crear una instancia del “JsonReader” este componente nos permitirá leer e interpretar la respuesta del servidor para cada petición Ajax realizada. Lo que tenemos que notar es el atributo “messageProperty” que define la propiedad que aloja un mensaje en la respuesta del servidor, otra propiedad importante es “successProperty” la cual indicará si la operación ha sido realizada con éxito.

En el paso dos colocamos los campos que conforman el “Record” que contendrá la información que usaremos en el grid.

En el paso tres se crea la instancia del “JsonWriter”, el atributo “encode” lo usamos para poner la información que se escribe en el grid en formato JSON, el atributo “writeAllFields” forza al “writer” a enviar todos los campos del record y no solamente aquellos que se han modificado.

Lo siguiente es hacer el store que tendrá al “JsonReader”, “JsonWriter” y al “HttpProxy”. Lo haremos de la siguiente manera:

```

1. this.storeGrid = new Ext.data.Store({ //step 1
2. id : "id",
3. proxy : proxy,
4. reader : reader,
5. writer : writer,
6. autoSave : true //<--hace las peticiones al servidor automáticamente
7. });
8.
9. this.storeGrid.load(); //step 2

```

En el paso uno lo que haces es hacer una instancia del store, esto es con el motivo de unir el “reader”, el “writer” y el “proxy”. Notemos que el “JsonWriter” es introducido al store de la misma manera de la que lo hacemos con el “JsonReader”, el store lo creamos como un objeto dentro de la función “init”.

En el paso dos cargamos el store como siempre lo hemos hecho, esto ejecuta una petición Ajax al servidor.

Con esto tenemos lo preparado al store para poder hacer las operaciones del CRUD de manera automática y muy sencilla.

### Creando un Grid Editable

A continuación crearemos el grid editable, lo primero que haremos es hacer el grid y desplegar la información que traemos desde la base de datos.

```
1. var textFieldEmail = new Ext.form.TextField({vtype: "email",allowBlank: false}), //step 1
2. textField = new Ext.form.TextField({allowBlank: false}),
3. sm = new Ext.grid.CheckboxSelectionModel();
4.
5. this.grid = new Ext.grid.EditorGridPanel({ //step 2
6. store : this.storeGrid, //step 3
7. columns : [
8. sm,
9. {header:'E-mail', dataIndex:'email', sortable: true, width:150, editor:textFieldEmail},
10. {header:'Name', dataIndex:'first', sortable: true, editor:textField},
11. {header:'Lastname', dataIndex:'last', sortable: true, editor:textField}
12.],
13. sm : sm,
14. border : false,
15. stripeRows : true
16. });

});
```

En el paso uno creamos lo necesario para poder hacer cambios a nuestra tabla, lo que hicimos fue crear una instancia de “TextField” con un vtype de tipo “email”, esto es con el fin de que cuando se intente hacer un cambio o crear un nuevo contacto se haga un validación de que lo que se introdujo sea una dirección de correo electrónico válida. Y por ultimo con el “allowBlank” no dejamos que el campo este vacío.

Se crea una segunda instancia de “TextField” pero esta no cuenta con un vtype ya que en esta no se tiene que hacer alguna validación de ese tipo pero si tiene el “allowBlank” para no permitir que este campo este vacío.

Por último creamos un “checkbox” para hacer el selectionModel. El resto del código es conocido para nosotros si no te recomendamos ver los tutoriales sobre esto.

En el paso dos creamos la tabla que editaremos.

En el paso tres le pasamos el “storeGrid” que contiene la información a desplegar en el grid.

Por último colocaremos nuestra tabla en una ventana para poder mostrar los registros de esta.

```
1. var win = new Ext.Window({
2. title : "CRUD Example",
3. layout : "fit",
4. tbar : [
5. {text:'New Contact', scope:this, handler:this.addContact,iconCls:'save-icon'},
6. {text:"Delete", scope:this, handler:this.onDelete,iconCls:'delete-icon'}],
7. width : 400,
8. height : 300,
9. items : [this.grid]
```

```

10. });
11. win.show();

```

Creamos la ventana esto ya es conocido para nosotros. Es importante mencionar que se crean los botones de “Nuevo Contacto” y “Eliminar” en la barra de herramienta, es importante mencionar que se les esta colocando una imagen a los botones, para esto es necesario definir las clases CSS (save-icon) y CSS(delete-icon) en nuestra hoja de estilos o documento HTML de la siguiente manera:

1. .save-icon{background:transparent url(Icons/add.png) 0 0 no-repeat !important;}
2. .delete-icon{background:transparent url(Icons/delete.png) 0 0 no-repeat !important;}

Por el momento no trabajaremos con las funciones que están en los botones ya que lo haremos un poco más adelante así que puedes comentar el atributo “handler”. En este momento deberíamos tener algo como lo siguiente imagen:



Al ejecutar nuestra aplicación podemos notar que los campos de la tabla tienen los registros que trajimos de la base de datos.

### Grabar los Registro Modificados

Hasta este punto tenemos la información en nuestro “Grid” lo siguiente que tenemos que hacer para poder cumplir una de las funciones del CRUD es editar los registros del “Grid” y guárdarlos en la base de datos.

En este momento si editamos un registro no se actualiza en la base de datos, pero si se cambia en el “Grid”. Al intentar hacer esto nos marca el siguiente error:

1. POST serverside/updateContact.php 404 Not Found

El error 404 nos indica que la url no existe, así que vamos a crear el archivo “serverside/updateContact.php”, este archivo es el que se conecta a la base de datos para poder actualizar el registro, a continuación veremos la estructura de este archivo.

```

1. <?php //step 1
2. $connection=mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection) or die("Error loading the DataBase".mysql_error());
4.
5. $info = $_POST["data"];
6.
7. $data = json_decode(stripslashes($info)); //step 2
8.
9. $mail = $data->email; //step 3
10. $name = $data->first;
11. $last = $data->last;
12. $id = $data->id;
13.

```

```

14. //step 4
15. $query = sprintf("UPDATE contacts SET email = '%s', firstName = '%s', lastName = '%s' WHERE id=%d",
16. mysql_real_escape_string($mail),
17. mysql_real_escape_string($name),
18. mysql_real_escape_string($last),
19. mysql_real_escape_string($id));
20.
21. $rs = mysql_query($query);
22.
23. echo json_encode(array(//step 5
24. "success" => mysql_errno() == 0,
25. "msg" => mysql_errno() == 0?"Contact inserted successfully":mysql_error()
26.));

```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos, una vez que recibimos la información que mando el “Store” lo decodificamos con la función `json_decode()` de php con esto ya podemos acceder a la información que fue editada en el “Grid”, la función “stripslashes” elimina el carácter “\” que Apache le agrega, es posible que la configuración de tu servidor no lo haga, pero es bueno prevenir esto para evitar errores en el futuro.

En el paso tres obtenemos la información y la separamos en variables para poder insertarlas a la base de datos, esto es opcional ya que podríamos utilizar el mismo objeto `$data`.

En el paso cuatro hacemos un “update” a la tabla en la que están guardados nuestros contactos y le damos como valores la información que fue mandada por el “Store”.

En el paso cinco lo que hacemos es mandar un mensaje de éxito si es que todo sale bien, de lo contrario se manda el mensaje de error de mysql.

Con esto si intentamos nuevamente hacer un cambio a un registro en el “Grid” veremos algo como lo siguiente:

| E-mail               | Name  | Lastname |
|----------------------|-------|----------|
| pedro_2010@gmail.com | Pedro | Lopez    |
| jose@hotmail.com     | Jose  | Paz      |
| john@hotmail.com     | John  | Smit     |

Actualizar un contacto

En este momento ya se guardan los registros en la base de datos y se tiene como respuesta algo como lo siguiente:

1. {"success":true,"msg":"Contact inserted successfully"}

Hasta este momento ya cumplimos con dos de las cuatro funciones de un CRUD leer y actualizar si notamos la mayor parte de esto fue manejado por el “proxy”, el “JsonReader” y el “JsonWriter” nosotros solo creamos los archivos php, esto nos facilita mucho las cosas.

## Crear un registro nuevo

En la sección anterior logramos guardar los registros que fueron modificados por el usuario, ahora vamos a insertar nuevos registros en el Grid, para esto usaremos la función (addContact) que fue asignada al botón “nuevo contacto” esta función nos ayudara a realizar lo antes mencionado.

```
1. addContact : function(){
2. var contact = new this.storeGrid.recordType({ //step 1
3. first : "",
4. last : "",
5. email : ""
6. });
7. //step 2
8. this.grid.stopEditing();
9. this.storeGrid.insert(0,contact);
10. this.grid.startEditing(0,1);
11. }
```

Creamos la función dentro del objeto principal y en el paso uno hacemos un nuevo registro poniendo los campos en del “Grid” vacíos para que después el usuario pueda introducir la información necesaria, también podríamos definir valores por defecto.

En el paso dos con “stopEditing” nos aseguramos que el usuario no esté modificando los campos, en el momento que se decida introducir un nuevo contacto. Ya que nos aseguramos que no se están editando, insertamos el nuevo contacto con los campos vacíos, y por último permitimos al usuario editar los campos con “startEditng(0,1)” notemos que éste recibe como parámetros la posición de la celda donde se inicia a editar donde “0” es el numero de fila y “1” es el número de columna.

Si ejecutamos nuestra aplicación veremos algo semejante a la siguiente imagen:

|                                     | E-mail               | Name      | Lastname |
|-------------------------------------|----------------------|-----------|----------|
| <input checked="" type="checkbox"/> | alejandro@gmail.com  | Alejandro | Cru      |
| <input type="checkbox"/>            | pedro_2010@gmail.com | Pedro     | Lopez    |
| <input type="checkbox"/>            | jose@hotmail.com     | Jose      | Paz      |
| <input type="checkbox"/>            | john@hotmail.com     | John      | Smit     |

Creando un contacto

Notemos que nos manda el siguiente error:

```
1. POST serverside/createContact.php 404 Not Found
```

Al igual que en el caso anterior tenemos que crear el archivo “createContact.php” en el cual se hacer la conexión a la base de datos y se guarda el nuevo registro, la estructura del archivo se muestra a continuación:

```
1. <?php
2. //step 1
3. $connection = mysql_connect("localhost", "root", "") or die("Connection Failed".mysql_error());
4. mysql_select_db("test", $connection) or die("Error loading the DataBase".mysql_error());
5.
6. $info = $_POST["data"];
7.
8. $data = json_decode(stripslashes($info)); //step 2
9.
```

```

10. $name = $data->first; //step 3
11. $last = $data->last;
12. $email = $data->email;
13.
14. //step 4
15. $query = sprintf("INSERT INTO contacts (email,firstName,lastName) values ('%s','%s','%s')",
16. mysql_real_escape_string($email),
17. mysql_real_escape_string($name),
18. mysql_real_escape_string($last));
19.
20. $rs = mysql_query($query);
21.
22. //step 5
23. echo json_encode(array(
24. "success" => mysql_errno() == 0,
25. "msg" => mysql_errno() == 0?"Contact inserted successfully":mysql_error(),
26. "data" => array(
27. array(
28. "id" => mysql_insert_id(),
29. "first" => $name,
30. "last" => $last,
31. "email" => $email
32.)
33.)
34.));

```

La estructura de este archivo es muy semejante a la de los archivos previos.

En el paso uno lo único que hacemos es hacer la conexión a la base de datos.

En el paso dos decodificamos la información que nos manda el “Store”, esto lo hacemos mediante la función Json\_decode() y le pasamos la información la cual es el registro que creó el usuario.

En el paso tres lo que hacemos es sacar esa información y guardarla en unas variables para poder insertar esa información a la tabla “contacts” de nuestra base de datos.

En el paso cuatro creamos el query que insertara los datos a la base de datos, es un query simple un “insert” le damos los campos y los valores que es la información que obtuvimos en el paso tres.

En el paso cinco lo que hacemos es mandar un mensaje y la información que fue insertada al base de datos.

Es muy importante mencionar que en el Json\_encode que se está haciendo después del “success” y el “msg” mandamos la información que fue insertada a la base de datos, es muy importante mandar el “id” del nuevo registro, esto nos permitirá que los métodos update y delete funcionen correctamente, si no lo hacemos no funcionarán.

Si intentamos insertar un nuevo registro esta será la respuesta:

1. {"success":true,"msg":"Contact inserted successfully","data":[{"id":4,"first":"Alejandro","last":"Cruz","email":"alejandro@gmail.com<script type="text/javascript">
2. /\* <![CDATA[ \*/
3. (function(){try{var s,a,i,j,r,c,l=document.getElementById("\_\_cf\_email\_\_");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}}catch(e){{}();}
4. /\* ]]> \*/
5. </script>"}]}

Con esto ya podremos insertar un nuevo registro en nuestro “Grid” y en nuestra base de datos.

## Eliminar un registro

En estos momentos solo hemos hecho lo necesario para poder leer los registros y desplegarlos en nuestro “Grid”, podemos añadir nuevo registros y editar los registros existentes. Lo único que nos hace falta es eliminar registros y con esto tendríamos las cuatro funciones del CRUD.

Para esto crearemos la función a la que hace referencia el botón “eliminar”, lo que haremos es lo siguiente:

```
1. onDelete : function(){ //step 1
2. var rows = this.grid.getSelectionModel().getSelections(); //step 2
3.
4. if(rows.length === 0){ //step 3
5. return false;
6. }
7. this.storeGrid.remove(rows); //step 4
8. },
```

En el paso uno creamos la función dentro del objeto principal.

En el paso dos tomamos la instancia del “SelectionModel()” de nuestro “Grid”, con esto podemos obtener la información de las filas seleccionadas por el usuario mediante el método “getSelections()”.

En el paso tres lo que hacemos es verificar que el usuario halla seleccionado algo, si no hay nada seleccionado o la selección del “Grid” esta vacía no se eliminará ningún registro ya que hacemos un return.

En el paso cuatro ya que se verifico que es un registro que puede ser borrado lo elimina del “Grid” usando el método “remove” del store que contiene los records.

Si intentamos hacer esto notaremos que efectivamente se borra del “Grid”, pero no se borra de nuestra base de datos, además nos manda el siguiente error:

```
1. POST serverside/destroyContact.php 404 Not Found
```

Este es el mismo mensaje que recibimos en las secciones anteriores, ya que aún no tenemos ese archivo PHP; este archivo se encarga de borrar el registro en la base de datos, es muy semejante a los que hemos hecho, su estructura es la siguiente:

```
1. <?php //step 1
2. $connection=mysql_connect("localhost","root","");
3. or die("Connection Failed".mysql_error());
4. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
5.
6. $id = json_decode(stripslashes($_POST["data"])); //step 2
7.
8. //step 3
9. $query = sprintf("DELETE FROM contacts WHERE id = %d",
10. mysql_real_escape_string($id));
11.
12. $rs = mysql_query($query);
13.
14. //step 4
15. echo json_encode(array(
16. "success" => mysql_errno() == 0,
17. "msg" => mysql_errno() == 0?"Contact deleted successfully":mysql_error()
18.));
```

En el paso uno lo que hacemos es crear la conexión a la base de datos.

En el paso dos obtenemos la información que nos fue mandada, la descodificamos con Json\_decode() y la guardamos en la variable “id”. En este caso solo se nos envió el “id” del registro que se quiere eliminar.

En el paso tres solo hacemos el query para eliminar el registro.

En el paso cuatro mandamos como respuesta un mensaje de éxito y si ocurrió un error se manda el error como respuesta.

Si intentamos eliminar un registro notaremos que esta vez también fue borrado de nuestra base de datos. Y esta sería la respuesta del servidor:

1. {"success":true,"msg":"Contact deleted successfully"}

Con esto terminamos la última parte de nuestro CRUD que es eliminar un registro.

## Conclusión

En esta ocasión vimos como podemos crear, leer, actualizar y eliminar datos usando “HttpProxy”, “JsonReader”, “JsonWriter” y un “Store” esto es muy útil ya que por medio de estas clases de Extjs no tenemos que hacer las peticiones al servidor manualmente como lo hemos hecho en tutoriales pasados. El “JsonReader” se encarga de leer la información que nos manda el servidor, mientras el “JsonWriter” codifica la información y hace la petición al servidor.

## Árboles con ExtJS

Los árboles son útiles para mostrar de forma visual una jerarquía de organización, por ejemplo un directorio de documentos, una familia o grupo de personas, etc. Ext proporciona el componente ideal para esta tarea.

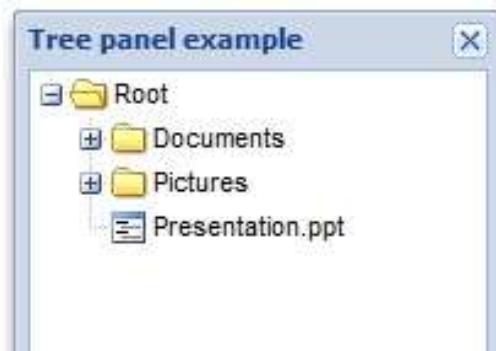
### El tree panel

veremos rápidamente como mostrar una estructura de árbol utilizando el TreePanel, es muy común usar este componente para mostrar un sistema de archivos, administrar un menú, administrar categorías y muchísimas aplicaciones que podemos realizar con este componente.

Una estructura de árbol es compleja de representarla gráficamente, sobre todo porque se complica dibujar correctamente los elementos o nodos del árbol, el componente TreePanel nos facilita notablemente este trabajo y nos permite crear aplicaciones con una interfaz rica.

### Demo

He preparado una demostración de lo que haremos al final de este tutorial, te invito a que la pruebes y veas el resultado que obtendremos.



El Tree Panel

## Material de apoyo

Para continuar es necesario descargar el material de apoyo y copiarlo dentro de nuestro servidor Web que estamos usando para este curso.

En el archivo JS encontramos el siguiente código:

1. Ext.ns('com.quizpot.tutorial');
- 2.

```

3. com.quizzpot.tutorial.TreeTutorial = {
4. init: function(){
5.
6. },
7.
8. //returns the data for the tree
9. getData: function(){
10.
11. }
12. }
13.
14. Ext.onReady(com.quizzpot.tutorial.TreeTutorial.init,com.quizzpot.tutorial.TreeTutorial);

```

A estas alturas del curso debemos estar familiarizados con el código anterior, simplemente se crea un espacio de nombres (namespace) para evitar problemas a futuro, dentro de la función “init” vamos a crear el tree y dentro de la función “getData” escribiremos la información que desplegará el componente.

### La información a desplegar

Lo más complejo al utilizar este componente es generar la información que desplegaremos, ya que la información es la clave para que funcione este componente adecuadamente, en esta ocasión la información la tendremos en memoria, decidí hacerlo así para explicar el formato en el que debemos generarla, en futuros tutoriales veremos cómo traer esta información dinámicamente desde el servidor usando Ajax, pero por ahora la tendremos en un objeto de JavaScript, el código siguiente debe ir dentro del método “getData”.

```

1. var root = {
2. text:'Root',
3. children:[
4. {
5. text:'Documents',
6. children:[{text:'file.doc',leaf:true}]
7. },
8. {
9. text:'Pictures',
10. children:[{text:'friends.jpg',leaf:true},{text:'working.jpg',leaf:true}]
11. },
12. {
13. text:'Presentation.ppt',
14. leaf:true
15. }
16.]
17. }
18.
19. return root;

```

Semejante a un árbol o planta (en el mundo real) el componente TreePanel también tiene una raíz (root), ramas (branches) y hojas (leaf) los cuales a su vez son nodos (Ext.treeTreeNode) , en el código anterior se está creando una estructura de árbol que inicia con la raíz, tiene dos ramas (documents y pictures) y cuatro hojas (file.doc, friends.jpg, working.jpg y presentation.ppt) distribuidas entre las ramas, a continuación voy a explicar las propiedades utilizadas en cada nodo:

- **text**, Esta propiedad es de suma importancia ya que define el “label” o texto que irá en el nodo, si no defines esta propiedad simplemente no se desplegará ningún texto para ese nodo.
- **children**: Esta propiedad es un arreglo de nodos, esto significa que el nodo actual es una rama del árbol (branch) y dentro de esta propiedad se encuentran más nodos los cuales a su vez pueden ser ramas u hojas, eso dependerá de tu información.
- **leaf**: Traducido al español significa “Hoja”, al configurar esta propiedad en “true” estas haciendo que el nodo se convierta en una hoja y por lo tanto no puede contener más nodos.

Por ahora con estas configuraciones basta, más adelante veremos algunas otras, como puedes ver crear los nodos es muy sencillo, la complejidad viene cuando esto se tiene que crear dinámicamente en el servidor, pero eso lo veremos en otro tutorial, por ahora dejémoslo así de sencillo.

## Creación del TreePanel

Esto es una tarea muy sencilla, y lo haremos en unas cuantas líneas de la siguiente manera:

```
1. var tree = new Ext.tree.TreePanel({ //step 1
2. border: false,
3. autoScroll:true,
4. root: this.getData() //step 2
5. });
6.
7. var win = new Ext.Window({ //step 3
8. title:'Tree panel example',
9. layout:'fit',
10. width:200,
11. height:300,
12. items: tree
13. });
14. win.show(); //step 4
```

En el paso 1 se crea el componente configurado para que no muestre el borde y para que nos ponga un scroll cuando el árbol crezca demasiado.

En el paso 2 es donde le asignamos la información que contendrá el componente, esta información la escribimos anteriormente en este tutorial.

En el paso 3 creamos la ventana que contendrá el árbol, ya debemos conocer esto pues lo hemos hecho en la mayoría de los tutoriales anteriores de este blog.

Finalmente en el paso 4 mostramos todo lo anterior.

## Expandiendo los nodos

En ocasiones se nos pide que el árbol aparezca expandido desde el principio, esto es muy sencillo simplemente tomamos el root y lo expandimos de la siguiente manera:

```
1. tree.getNode().expand();
```

También podemos agregar en la configuración del "root" la propiedad "expanded" y el nodo aparecerá expandido automáticamente:

```
1. var root = {
2. text:'Root',
3. expanded: true, // <--- aparece expandido el root
4. children:[
5. {
6. text:'Documents',
7. children:[{text:'file.doc',leaf:true}]
8. },
9. {
10. text:'Pictures',
11. children:[{text:'friends.jpg',leaf:true},{text:'working.jpg',leaf:true}]
12. },
13. {
14. text:'Presentation.ppt',
15. leaf:true
16. }
17.]
18. }
```

Esta es una nueva propiedad que nos permite configurar el nodo para que se muestre expandido desde el principio, es muy útil en algunas ocasiones.

## Conclusiones

En éste tutorial vimos como crear un árbol de una manera muy sencilla, también aprendimos como expandirlo de dos maneras diferentes y la importancia que tiene la información que se desplegará, realmente mayor el trabajo de mostrar un árbol lo hacemos en el servidor para generar dinámicamente la estructura, en tutoriales siguientes veremos algunas maneras de hacer esto.

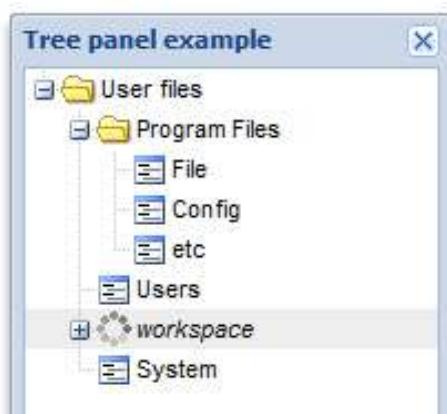
## Árboles generados con Ajax

En el tutorial anterior vimos como crear un TreePanel utilizando información alojada en un arreglo de JavaScript, el día de hoy veremos cómo generar el árbol utilizando Ajax para cargar el contenido dinámicamente.

En sistemas grandes que manejan mucha información es recomendable mostrarla poco a poco para no sobrecargar al explorador y nuestra aplicación se ejecute más rápido, utilizando Ajax podemos ir expandiendo el árbol que necesitamos mostrar de una manera muy sencilla, en este tutorial veremos como hacerlo.

### Demostración

He preparado una demostración de lo que haremos en este tutorial, te recomiendo le des un vistazo para que tengas una idea del resultado final.



TreePanel generado con Ajax

### Material de apoyo

Antes de continuar es necesario descargar el material de apoyo y copiarlo dentro de nuestro servidor Web que instalamos al inicio del curso.

El material consta de tres archivos, un PHP que contiene la información en un arreglo, un HTML que solamente importa el Framework de ExtJS y un JS donde escribiremos el código para este tutorial, es importante mencionar que este último archivo solamente tiene definido el “namespace” que utilizaremos.

### Crear el TreePanel

Vamos a crear el componente que alojará nuestro árbol, el código siguiente debe ir dentro de la función “init” que viene en el JS del material de apoyo:

```
1. var loader = new Ext.tree.TreeLoader({ //Paso 1
2. url: 'tree-ajax.php'
3. });
4.
5. var tree = new Ext.tree.TreePanel({ //Paso 2
6. border: false,
7. autoScroll:true,
8. //dataUrl:'tree-ajax.php' //<--- Así nos crea automáticamente el TreeLoader
9. loader:loader //para fines didácticos he creado el TreeLoader a mano
10. });
11.
```

```

12. var root = new Ext.tree.AsyncTreeNode({ //Paso 3
13. text: 'User files'
14. });
15.
16. tree.setRootNode(root); //Paso 4

```

En el paso 1 hemos creado un TreeLoader, en el objeto de configuración solamente definimos la propiedad “url” a donde se realizará la llamada Ajax para ir cargando los nodos.

En el paso 2 creamos el TreePanel, y le agregamos el “loader” que definimos en el paso anterior, esta es la única diferencia con respecto al tutorial anterior.

En el paso 3 creamos el nodo raíz, este nodo es el que contendrá todas las ramas y hojas que cargaremos mediante Ajax, puedes ver que se ha creado una instancia del componente “Ext.tree.AsyncTreeNode” el cual cargará los nodos dinámicamente utilizando el “loader” que ya configuramos anteriormente.

El paso 4 es de suma importancia, aquí estamos asignándole al Tree su raíz, si no hacemos este paso simplemente no veremos nada.

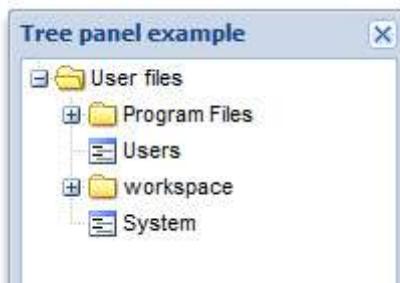
### **Mostrando el árbol**

Hasta ahora solamente tenemos en memoria nuestro árbol, falta “renderizarlo” para que se pueda visualizar en la pantalla, esto lo haremos asignándoselo a una ventana de la siguiente manera:

```

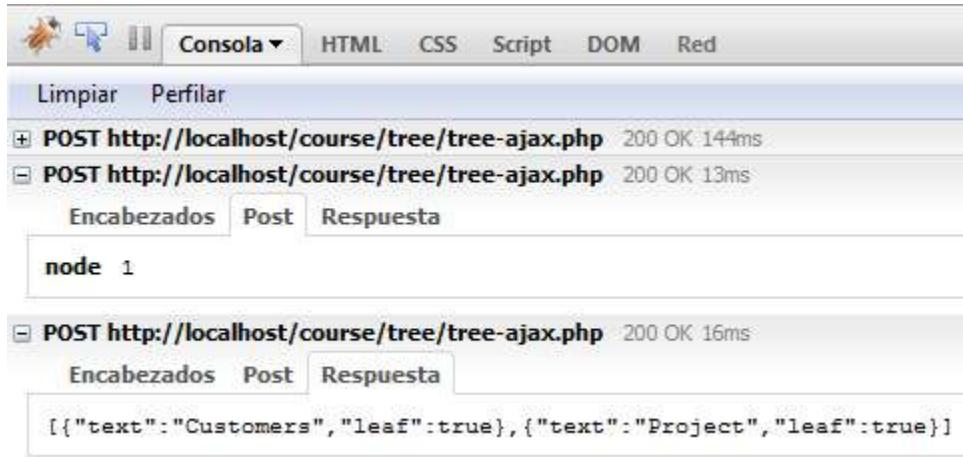
1. var win = new Ext.Window({
2. title:'Tree panel example',
3. layout:'fit',
4. width:200,
5. height:300,
6. items: tree
7. });
8. win.show();

```



TreePanel dinámico

La siguiente imagen muestra las peticiones Ajax que se han hecho cada que expandimos un nodo.



## Peticiones Ajax

### La información en el servidor

Debemos recordar que la información arrojada por el servidor debe estar en formato JSON con las propiedades definidas en el tutorial anterior, el PHP que viene con el material de apoyo es el siguiente.

```

1. <?php
2.
3. $id = $_POST['node']; //Paso 1
4.
5. $nodes = array(//Paso 2
6. array(
7. 'id'=>1,
8. 'text'=>'Program Files',
9. 'files'=>array(
10. array(
11. 'text'=>'File',
12. 'leaf'=>true
13.),
14. array(
15. 'text'=>'Config',
16. 'leaf'=>true
17.),
18. array(
19. 'text'=>'etc',
20. 'leaf'=>true
21.)
22.)
23.),
24. array(
25. 'id'=>2,
26. 'text'=>'Users',
27. 'leaf'=>true
28.),
29. array(
30. 'id'=>3,
31. 'text'=>'workspace',
32. 'files'=>array(
33. array(
34. 'text'=>'Customers',
35. 'leaf'=>true
36.),
37. array(
38. 'text'=>'Project',

```

```

39. 'leaf'=>true
40.)
41.)
42.),
43. array(
44. 'id'=>4,
45. 'text'=>'System',
46. 'leaf'=>true
47.)
48.);
49.
50. if(is_numeric($id)){ //Paso 3
51. $nodes = isset($nodes[$id-1]['files'])?$nodes[$id-1]['files']:array();
52. }
53.
54. echo json_encode($nodes); //Paso 4
55. ?>

```

En el paso 1 tomamos el “id” del nodo que ha sido solicitado, esto es importante para que podamos regresar la información correcta.

En el paso 2 creamos una pequeña “Base de datos”, simplemente es la información que desplegaremos, puedes ver que he definido una propiedad llama “files”, esta propiedad la asigné por conveniencia ya que si la hubiera nombrado “children” el componente la renderizará desde el principio.

En el paso 3 verificamos que el “id” sea numérico, de esta manera nos aseguramos que podremos buscar en el arreglo principal la información que nos ha sido solicitada.

En el paso 4 solamente se imprime en formato JSON los nodos a mostrar en el Tree.

### **Conclusiones**

Crear un árbol con Ajax es muy sencillo, lo complicado es generar la información que desplegaremos, en futuros tutoriales mostraré como integrarlo con una base de datos.

## **Cambiar el ícono a los nodos de un TreePanel**

El día de hoy veremos cómo cambiar el ícono de los nodos a un TreePanel, esto es algo muy útil para poder representar sistemas de archivos, organigramas, categorías, etc.

Para este tutorial representaremos un organigrama empresarial, cambiaremos los iconos de cada nodo para lograr nuestro objetivo. He preparado una demostración para que puedas ver el resultado final.



Demostración del tutorial

### Material de apoyo

Vamos a descargar el material de apoyo el cual consta de un archivo HTML donde únicamente estamos importando la librería de ExtJS, también cuenta con un JS el cual está en blanco y sobre el cual escribiremos el código de este tutorial, también encontrarás una carpeta con los iconos que usaremos.

Recuerda copiar todos los archivos al servidor Web que instalamos al inicio de este curso.

### Namespace

Vamos a encapsular nuestro código dentro de un buen namespace.

```

1. Ext.ns("com.quizzpot.tutorial");
2.
3. com.quizzpot.tutorial.TreeIconsTutorial = {
4. init: function(){
5. //code goes here
6. },
7.
8. getData: function(){
9. //here we are going to define the data
10. }
11. }
12.
13. Ext.onReady(com.quizzpot.tutorial.TreeIconsTutorial.init,com.quizzpot.tutorial.TreeIconsTutorial);

```

He creado dos funciones dentro del objeto principal, si has venido siguiendo el curso sabrás que dentro de la función “init” escribiremos el código principal de nuestra aplicación.

Dentro de la función “getData” escribiremos el código necesario para generar la información que desplegaremos en el TreePanel, ya hemos hecho algo semejante en el tutorial anterior.

### Creación del TreePanel

Crear el TreePanel es algo muy sencillo y lo realizamos en unas pocas líneas de código de la siguiente manera:

```
1. var tree = new Ext.tree.TreePanel({
2. border: false,
3. autoScroll:true,
4. rootVisible:false,
5. root: this.getData()
6. });
```

Con respecto al tutorial anterior la única diferencia es que he agregado la propiedad “rootVisible” la cual nos permite mostrar u ocultar el nodo “root”, por defecto esta como visible, pero para este ejemplo he decidido ocultarlo.

### Mostrar el Tree en una ventana

Hasta ahora el Tree solamente existe en memoria, necesitamos renderizarlo en nuestra página Web para que el usuario pueda verlo, existen varias maneras de hacerlo, para este ejemplo usaremos una ventana que contenga al Tree.

```
1. var win = new Ext.Window({
2. title:'Icons example',
3. layout:'fit',
4. width:300,
5. height:400,
6. items: tree
7. });
8. win.show();
```

Si en este momento actualizamos el explorador nos aparecerá un error en el Firebug mostrando “this.root is undefined”, esto ocurre porque no hemos definido el “root”, para definirlo vamos a modificar la función “getData” de la siguiente manera:

```
1. getData: function(){
2. //here we are going to define the data
3. return {
4. text:'The root node'
5. }
6. }
```



Ventana con TreePanel vacío

Nota que el root está oculto y como no tenemos información extra simplemente aparece en blanco la ventana.

### Definiendo el organigrama

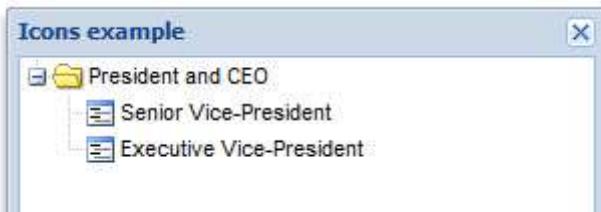
Cuando usamos el componente TreePanel debemos tener en cuenta que la información es lo más importante ya que le da vida a este componente, vamos a definir algunas posiciones o puestos en el organigrama que desplegaremos, esto lo hacemos en la función “getData”.

```
1. getData: function(){
2. //here we are going to define the data
3. return {
4. text:'The root node',
```

```

5. children:[
6. {
7. text:'President and CEO',
8. children:[
9. {
10. text:'Senior Vice-President',
11. leaf:true
12. },
13. text:'Executive Vice-President',
14. leaf:true
15. }
16.]
17. }
18.]
19. }
20. }
```

Hasta aquí nada nuevo, ya hemos hablado de esta estructura en el tutorial anterior, si actualizamos nuestro navegador veremos algo como la siguiente imagen:



TreePanel con sus iconos por defecto

### Cambiando los íconos

Una vez construido lo anterior llegamos a la parte principal de este tutorial, vamos a cambiar los iconos de cada nodo para darle una mejor apariencia.

```

1. return {
2. text:'The root node',
3. children:[
4. {
5. text:'President and CEO',
6. iconCls:'ceo-icon', //the icon CSS class
7. children:[
8. {
9. text:'Senior Vice-President',
10. iconCls:'vice-icon',//the icon CSS class
11. leaf:true
12. },
13. text:'Executive Vice-President',
14. iconCls:'vice-icon',//the icon CSS class
15. leaf:true
16. }
17.]
18. }
19.]
20. }
```

Hemos utilizado la propiedad "iconCls" para asignarle una clase CSS en la cual tenemos definido el icono que usaremos, lo siguiente que debemos hacer es definir esas clases CSS en nuestro documento HTML o en nuestra hoja de estilos de la siguiente manera:

1. .ceo-icon{background:transparent url(icons/user\_gray.png) 0 0 no-repeat !important;}
2. .vice-icon{background:transparent url(icons/user\_suit.png) 0 0 no-repeat !important;}

Nótese que se está haciendo uso de “!important” en la definición de los estilos.

Si actualizamos el navegador veremos algo semejante a la siguiente imagen.



TreePanel con íconos personalizado

A continuación voy a definir más puestos en el organigrama, es algo muy sencillo y que podríamos hacerlo dinámicamente en el servidor y desplegarlo en el TreePanel usando Ajax para actualizarlo, pero para hacerlo más rápido decidí hacerlo de esta manera.

```
1. var root = {
2. text:'Root',
3. children:[
4. {
5. text:'CEO',
6. iconCls:'ceo-icon',
7. children:[
8. {
9. text:'Assistant',
10. iconCls:'assistant-icon',
11. leaf:true
12. },
13. {
14. text:'Software Architect',
15. iconCls:'architect-icon',
16. leaf:true
17. },
18. {
19. text:'Project Manager',
20. iconCls:'pm-icon',
21. children:[
22. {
23. text:'Developers team leader',
24. iconCls:'developer-icon',
25. children:[
26. {
27. text:'Software developer',
28. iconCls:'developer-icon',
29. leaf:true
30. },
31. {
32. text:'Software developer',
33. iconCls:'developer-icon',
34. leaf:true
35. },
36. {
37. text:'Software developer',
38. iconCls:'developer-icon',
39. leaf:true
40. },
41. {
42. text:'Software developer',
43. iconCls:'developer-icon',
44. leaf:true
45. }
46. }
47. }
48. }
49. }
50. }
51. }
52. }
53. }
54. }
55. }
56. }
57. }
58. }
59. }
60. }
61. }
62. }
63. }
64. }
65. }
66. }
67. }
68. }
69. }
70. }
71. }
72. }
73. }
74. }
75. }
76. }
77. }
78. }
79. }
80. }
81. }
82. }
83. }
84. }
85. }
86. }
87. }
88. }
89. }
90. }
91. }
92. }
93. }
94. }
95. }
96. }
97. }
98. }
99. }
100. }
101. }
102. }
103. }
104. }
105. }
106. }
107. }
108. }
109. }
110. }
111. }
112. }
113. }
114. }
115. }
116. }
117. }
118. }
119. }
120. }
121. }
122. }
123. }
124. }
125. }
126. }
127. }
128. }
129. }
130. }
131. }
132. }
133. }
134. }
135. }
136. }
137. }
138. }
139. }
140. }
141. }
142. }
143. }
144. }
145. }
146. }
147. }
148. }
149. }
150. }
151. }
152. }
153. }
154. }
155. }
156. }
157. }
158. }
159. }
160. }
161. }
162. }
163. }
164. }
165. }
166. }
167. }
168. }
169. }
170. }
171. }
172. }
173. }
174. }
175. }
176. }
177. }
178. }
179. }
180. }
181. }
182. }
183. }
184. }
185. }
186. }
187. }
188. }
189. }
190. }
191. }
192. }
193. }
194. }
195. }
196. }
197. }
198. }
199. }
200. }
201. }
202. }
203. }
204. }
205. }
206. }
207. }
208. }
209. }
210. }
211. }
212. }
213. }
214. }
215. }
216. }
217. }
218. }
219. }
220. }
221. }
222. }
223. }
224. }
225. }
226. }
227. }
228. }
229. }
230. }
231. }
232. }
233. }
234. }
235. }
236. }
237. }
238. }
239. }
240. }
241. }
242. }
243. }
244. }
245. }
246. }
247. }
248. }
249. }
250. }
251. }
252. }
253. }
254. }
255. }
256. }
257. }
258. }
259. }
260. }
261. }
262. }
263. }
264. }
265. }
266. }
267. }
268. }
269. }
270. }
271. }
272. }
273. }
274. }
275. }
276. }
277. }
278. }
279. }
280. }
281. }
282. }
283. }
284. }
285. }
286. }
287. }
288. }
289. }
290. }
291. }
292. }
293. }
294. }
295. }
296. }
297. }
298. }
299. }
300. }
301. }
302. }
303. }
304. }
305. }
306. }
307. }
308. }
309. }
310. }
311. }
312. }
313. }
314. }
315. }
316. }
317. }
318. }
319. }
320. }
321. }
322. }
323. }
324. }
325. }
326. }
327. }
328. }
329. }
330. }
331. }
332. }
333. }
334. }
335. }
336. }
337. }
338. }
339. }
340. }
341. }
342. }
343. }
344. }
345. }
346. }
347. }
348. }
349. }
350. }
351. }
352. }
353. }
354. }
355. }
356. }
357. }
358. }
359. }
360. }
361. }
362. }
363. }
364. }
365. }
366. }
367. }
368. }
369. }
370. }
371. }
372. }
373. }
374. }
375. }
376. }
377. }
378. }
379. }
380. }
381. }
382. }
383. }
384. }
385. }
386. }
387. }
388. }
389. }
390. }
391. }
392. }
393. }
394. }
395. }
396. }
397. }
398. }
399. }
400. }
401. }
402. }
403. }
404. }
405. }
406. }
407. }
408. }
409. }
410. }
411. }
412. }
413. }
414. }
415. }
416. }
417. }
418. }
419. }
420. }
421. }
422. }
423. }
424. }
425. }
426. }
427. }
428. }
429. }
430. }
431. }
432. }
433. }
434. }
435. }
436. }
437. }
438. }
439. }
440. }
441. }
442. }
443. }
444. }
445. }
446. }
447. }
448. }
449. }
450. }
451. }
452. }
453. }
454. }
455. }
456. }
457. }
458. }
459. }
460. }
461. }
462. }
463. }
464. }
465. }
466. }
467. }
468. }
469. }
470. }
471. }
472. }
473. }
474. }
475. }
476. }
477. }
478. }
479. }
480. }
481. }
482. }
483. }
484. }
485. }
486. }
487. }
488. }
489. }
490. }
491. }
492. }
493. }
494. }
495. }
496. }
497. }
498. }
499. }
500. }
501. }
502. }
503. }
504. }
505. }
506. }
507. }
508. }
509. }
510. }
511. }
512. }
513. }
514. }
515. }
516. }
517. }
518. }
519. }
520. }
521. }
522. }
523. }
524. }
525. }
526. }
527. }
528. }
529. }
530. }
531. }
532. }
533. }
534. }
535. }
536. }
537. }
538. }
539. }
540. }
541. }
542. }
543. }
544. }
545. }
546. }
547. }
548. }
549. }
550. }
551. }
552. }
553. }
554. }
555. }
556. }
557. }
558. }
559. }
560. }
561. }
562. }
563. }
564. }
565. }
566. }
567. }
568. }
569. }
570. }
571. }
572. }
573. }
574. }
575. }
576. }
577. }
578. }
579. }
580. }
581. }
582. }
583. }
584. }
585. }
586. }
587. }
588. }
589. }
590. }
591. }
592. }
593. }
594. }
595. }
596. }
597. }
598. }
599. }
600. }
601. }
602. }
603. }
604. }
605. }
606. }
607. }
608. }
609. }
610. }
611. }
612. }
613. }
614. }
615. }
616. }
617. }
618. }
619. }
620. }
621. }
622. }
623. }
624. }
625. }
626. }
627. }
628. }
629. }
630. }
631. }
632. }
633. }
634. }
635. }
636. }
637. }
638. }
639. }
640. }
641. }
642. }
643. }
644. }
645. }
646. }
647. }
648. }
649. }
650. }
651. }
652. }
653. }
654. }
655. }
656. }
657. }
658. }
659. }
660. }
661. }
662. }
663. }
664. }
665. }
666. }
667. }
668. }
669. }
670. }
671. }
672. }
673. }
674. }
675. }
676. }
677. }
678. }
679. }
680. }
681. }
682. }
683. }
684. }
685. }
686. }
687. }
688. }
689. }
690. }
691. }
692. }
693. }
694. }
695. }
696. }
697. }
698. }
699. }
700. }
701. }
702. }
703. }
704. }
705. }
706. }
707. }
708. }
709. }
710. }
711. }
712. }
713. }
714. }
715. }
716. }
717. }
718. }
719. }
720. }
721. }
722. }
723. }
724. }
725. }
726. }
727. }
728. }
729. }
730. }
731. }
732. }
733. }
734. }
735. }
736. }
737. }
738. }
739. }
740. }
741. }
742. }
743. }
744. }
745. }
746. }
747. }
748. }
749. }
750. }
751. }
752. }
753. }
754. }
755. }
756. }
757. }
758. }
759. }
760. }
761. }
762. }
763. }
764. }
765. }
766. }
767. }
768. }
769. }
770. }
771. }
772. }
773. }
774. }
775. }
776. }
777. }
778. }
779. }
780. }
781. }
782. }
783. }
784. }
785. }
786. }
787. }
788. }
789. }
790. }
791. }
792. }
793. }
794. }
795. }
796. }
797. }
798. }
799. }
800. }
801. }
802. }
803. }
804. }
805. }
806. }
807. }
808. }
809. }
810. }
811. }
812. }
813. }
814. }
815. }
816. }
817. }
818. }
819. }
820. }
821. }
822. }
823. }
824. }
825. }
826. }
827. }
828. }
829. }
830. }
831. }
832. }
833. }
834. }
835. }
836. }
837. }
838. }
839. }
840. }
841. }
842. }
843. }
844. }
845. }
846. }
847. }
848. }
849. }
850. }
851. }
852. }
853. }
854. }
855. }
856. }
857. }
858. }
859. }
860. }
861. }
862. }
863. }
864. }
865. }
866. }
867. }
868. }
869. }
870. }
871. }
872. }
873. }
874. }
875. }
876. }
877. }
878. }
879. }
880. }
881. }
882. }
883. }
884. }
885. }
886. }
887. }
888. }
889. }
890. }
891. }
892. }
893. }
894. }
895. }
896. }
897. }
898. }
899. }
900. }
901. }
902. }
903. }
904. }
905. }
906. }
907. }
908. }
909. }
910. }
911. }
912. }
913. }
914. }
915. }
916. }
917. }
918. }
919. }
920. }
921. }
922. }
923. }
924. }
925. }
926. }
927. }
928. }
929. }
930. }
931. }
932. }
933. }
934. }
935. }
936. }
937. }
938. }
939. }
940. }
941. }
942. }
943. }
944. }
945. }
946. }
947. }
948. }
949. }
950. }
951. }
952. }
953. }
954. }
955. }
956. }
957. }
958. }
959. }
960. }
961. }
962. }
963. }
964. }
965. }
966. }
967. }
968. }
969. }
970. }
971. }
972. }
973. }
974. }
975. }
976. }
977. }
978. }
979. }
980. }
981. }
982. }
983. }
984. }
985. }
986. }
987. }
988. }
989. }
990. }
991. }
992. }
993. }
994. }
995. }
996. }
997. }
998. }
999. }
1000. }
```

```

43.]
44. },
45. text:'QA team leader',
46. iconCls:'tester-icon',
47. children:[
48. {
49. text:'Tester',
50. iconCls:'tester-icon',
51. leaf:true
52. },
53. {
54. text:'Tester',
55. iconCls:'tester-icon',
56. leaf:true
57. },
58. {
59. text:'Tester',
60. iconCls:'tester-icon',
61. leaf:true
62. },
63. {
64. text:'Tester',
65. iconCls:'tester-icon',
66. leaf:true
67. },
68. {
69. text:'Tester',
70. iconCls:'tester-icon',
71. leaf:true
72. }
73.],
74. text:'Sales manager',
75. iconCls:'sales-icon',
76. leaf:true
77. },
78.],
79.},
80.]
81.};

```

Los nuevos estilos para los iconos son:

1. .manager-icon{background:transparent url(/icons/user\_orange.png) 0 0 no-repeat !important;}
2. .hc-icon{background:transparent url(/icons/user\_female.png) 0 0 no-repeat !important;}
3. .pm-icon{background:transparent url(/icons/user\_red.png) 0 0 no-repeat !important;}
4. .developer-icon{background:transparent url(/icons/user.png) 0 0 no-repeat !important;}
5. .tester-icon{background:transparent url(/icons/user\_green.png) 0 0 no-repeat !important;}

Con las modificaciones anteriores tenemos algo como la siguiente imagen.



Nuestro tutorial terminado lucirá así

### Conclusiones

Los TreePanel son muy útiles para desplegar información en forma de árbol, además es un componente muy flexible. El día de hoy solo vimos como cambiar el ícono de cada nodo, con esto podremos desarrollar interfaces muy atractivas y usables.

### Desplegar el sistema de archivos en un TreePanel

He recibido muchos mensajes solicitándome el código necesario para poder leer el sistema de archivos en el servidor e integrarlo a un TreePanel para que el usuario pueda administrar sus archivos, en este tutorial explico paso a paso como lograr esta integración.

El día de hoy veremos cómo leer el contenido de una carpeta en el servidor (utilizando PHP) para luego poder desplegar esa información en un TreePanel utilizando Ajax para ir cargando dinámicamente el contenido de las carpetas.



Desplegado el sistema de archivos en un TreePanel

### **Material de apoyo**

Para continuar es necesario descargar el material de apoyo, el contiene unos íconos y algunos archivos que desplegaremos en el TreePanel, es necesario que copies todos estos archivos a un servidor Web y cambiar la ruta de la librería ExtJS si es necesario.

### **Leyendo el contenido de una carpeta**

Voy a utilizar PHP para leer el contenido de la carpeta “files” la cual viene con el material de apoyo, dentro de esta carpeta existen varios archivos los cuales serán desplegados en un TreePanel. Vamos a editar el archivo “get-files.php” (viene en el material de apoyo) y escribiremos lo siguiente:

1. \$root = 'files/'; //step 1
2. \$node = isset(\$\_REQUEST['node'])?\$\_REQUEST['node']:""; //step 2

En el paso uno se define la ruta de la carpeta que será expuesta al usuario, aquí podemos definir cualquier carpeta que necesitemos exponer, por ejemplo “/home/john/docs” o también “c:\users\john\docs”, para este ejemplo usaremos “files/” la cual debe estar en el mismo nivel que nuestro archivo “get-files.php”.

En el segundo paso se está tomando el parámetro “node”, este parámetro contendrá la ruta dentro de la carpeta “files” que deseamos desplegar, recordemos que usaremos Ajax para ir cargando dinámicamente el contenido de cada carpeta conforme el usuario vaya navegando, esto es importante porque de esta manera haremos que nuestra aplicación solo cargue la información que el usuario solicita y por lo tanto mejoraremos el rendimiento.

Ahora escribiremos lo siguiente:

1. if(strpos(\$node, '..') !== false){
2. die('Nice try buddy.');
3. }

¡El código anterior es importantísimo! Ya que busca dentro de la variable “node” que no exista la secuencia de caracteres “..” ya que de existir un usuario malintencionado podría leer archivos que no debería poder ver, por ejemplo las credenciales de nuestra base de datos o cualquier otro archivo importante.

Una vez que nos aseguramos que el parámetro “node” viene limpio nos preparamos para leer los archivos de la ruta que el usuario ha solicitado.

```

1. $nodes = array();
2. $d = dir($root.$node); //step 1
3. while($f = $d->read()){ //step 2
4. if($f == '.' || $f == '..' || substr($f, 0, 1) == '.') continue; //step 3
5.
6. if(is_dir($root.$node.'/'.$f)){ //step 4
7. array_push($nodes,array('text'=>$f, 'id'=>$node.'/'.$f));
8. }else{
9. array_push($nodes, array('text'=>$f, 'id'=>$node.'/'.$f, 'leaf'=>true,'iconCls'=>getIcon($f)));
10. }
11. }
12. $d->close(); //step 5

```

En el paso uno usamos la función “dir” para abrir el directorio solicitado, primero asignamos la ruta principal y luego le concatenamos la ruta que el usuario solicitó.

En el paso dos mediante un ciclo “while” leemos todos los archivos contenidos en el directorio que abrimos en el paso uno y colocamos cada archivo en la variable “f”.

El paso tres es necesario para que no mostremos los directorios o archivos ocultos.

En el paso cuatro verificamos que estamos agregándole al arreglo “nodes” la información de cada archivo, para eso primero verificamos que tipo de archivo es, si es un directorio o carpeta solamente le asignamos el texto que desplegará y el identificador que tendrá, puedes ver que como ID le asignamos la ruta que enviará al servidor, por otro lado, si no es un directorio le asignamos la propiedad “leaf:true” al nodo que estamos creando y también le asignamos la clase CSS que contendrá al ícono correspondiente.

En el paso cinco cerramos el directorio que leímos anteriormente.

Ahora necesitamos definir la función “getIcon” de la siguiente manera:

```

1. function getIcon($name){
2. if (preg_match("^\\.png$/", $name) || preg_match("^\\.jpg$/", $name) || preg_match("^\\.gif$/", $name)) {
3. return 'jpg-icon';
4. }else if (preg_match("^\\.xls$/", $name) || preg_match("^\\.xlsx$/", $name)) {
5. return 'xls-icon';
6. }else if (preg_match("^\\.ppt$/", $name) || preg_match("^\\.pptx$/", $name)) {
7. return 'ppt-icon';
8. }else if (preg_match("^\\.doc$/", $name) || preg_match("^\\.docx$/", $name)) {
9. return 'doc-icon';
10. }else{
11. return 'unknow-icon';
12. }
13. }

```

Esta función recibe el nombre de un archivo y mediante una condición verificamos la terminación o “extensión” para poder asignarle el ícono correspondiente, si no encuentra ninguna extensión le asignaremos un ícono genérico.

En el material de apoyo he creado las clases CSS necesarias, ya hemos hablado de cómo cambiar los íconos a los nodos anteriormente.

```

1. .xls-icon{background:transparent url(Icons/page_excel.png) 0 0 no-repeat !important;}
2. .ppt-icon{background:transparent url(Icons/page_white_powerpoint.png) 0 0 no-
repeat !important;}
3. .doc-icon{background:transparent url(Icons/page_word.png) 0 0 no-repeat !important;}
4. .jpg-icon{background:transparent url(Icons/picture.png) 0 0 no-repeat !important;}

```

```
5. .unknow-icon{background:transparent url(icons/page_white.png) 0 0 no-repeat !important;}
```

Por último necesitamos generar el JSON con la información que hemos obtenido.

```
1. //print the data
2. echo json_encode($nodes);
```

Si ejecutas el archivo “get-files.php” en tu navegador debería regresarte la siguiente información:

```
1. [{"text":"docs","id":"Vdocs"}, {"text":"music","id":"Vmusic"}, {"text":"pictures","id":"Vpictures"}]
```

Recordemos que cuando utilizamos el componente TreePanel lo más importante es la información que éste contiene, hasta ahora solamente hemos creado la parte dinámica del nuestro ejemplo, pasemos a crear la parte visual.

### Creación del TreePanel

Hemos estudiado anteriormente como crear un TreePanel, realmente no tiene nada complicado ya que la parte complicada ya la hemos hecho y ahora si estamos listos para escribir el siguiente código dentro del archivo “tree-files.js”:

```
1. var tree = new Ext.tree.TreePanel({
2. border: false,
3. autoScroll:true,
4. dataUrl:'get-files.php',
5. root: new Ext.tree.AsyncTreeNode({
6. id:'.',
7. text: 'User files'
8. })
9. });
```

Lo más importante aquí es el identificador que le asigné al nodo “root”, el ID es el parámetro que por defecto el TreePanel envía al servidor cuando se hace una petición Ajax, por lo tanto el primer directorio que necesito que sea desplegado es el principal, por esta razón le asigné un punto ya que la ruta generada en el servidor es la siguiente para este caso:

```
1. // root = files/
2. // node = .
3. //root+node = files/.
4. $d = dir($root.$node);
```

Lo siguiente es renderizar el TreePanel en la pantalla, para esto usaré una ventana de la siguiente manera:

```
1. var win = new Ext.Window({
2. title:'Reading files example',
3. layout:'fit',
4. width:300,
5. height:400,
6. items: tree
7. });
8. win.show();
```



Pantalla inicial, todavía no cargamos la información hasta que el usuario la solicite

Ahora, si damos clic sobre cualquier carpeta automáticamente se realizará una llamada Ajax al servidor y este nos regresará los archivos necesarios.



Haciendo la petición Ajax



Desplegado los nuevos archivos

### Conclusiones

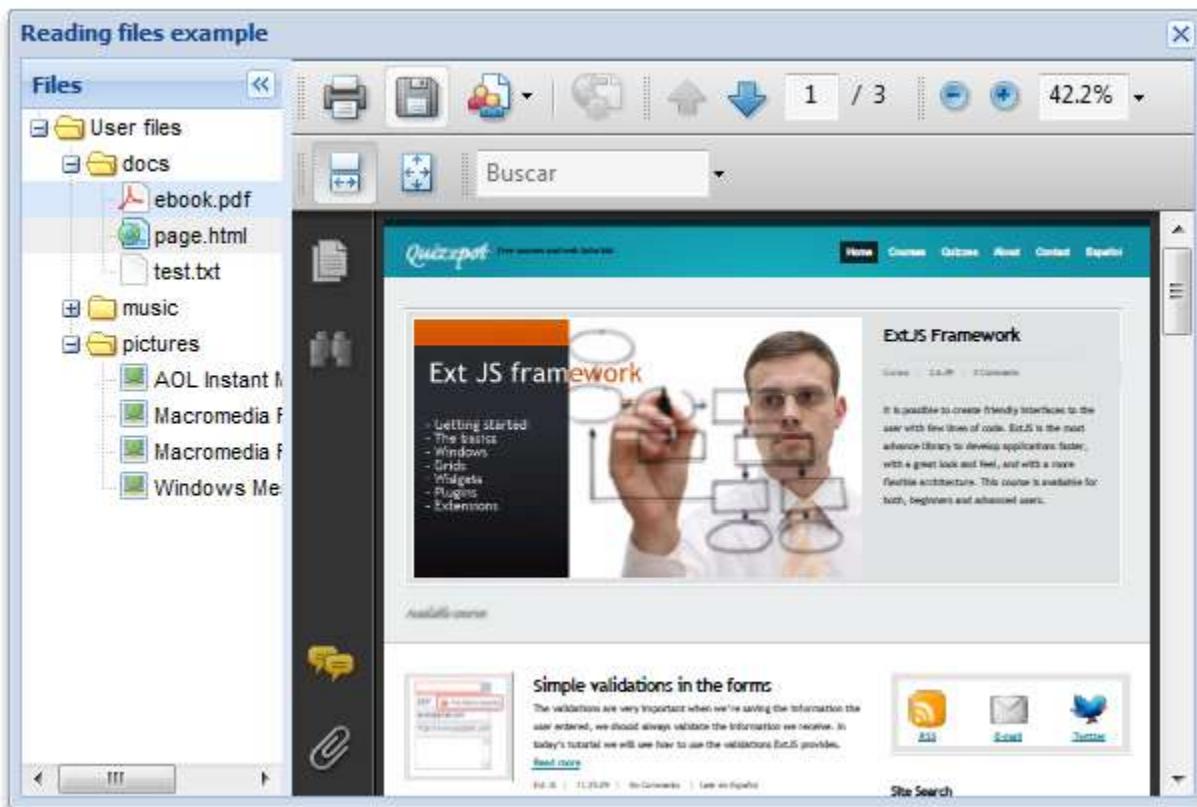
Hemos visto como exponer el contenido de una carpeta ubicada en nuestro servidor, utilizar el componente TreePanel es muy sencillo, más adelante veremos como interactuar con este componente, agregaremos algunos eventos y permitiremos que el usuario pueda mover los nodos de un lugar a otro.

## Explorador de archivos para el servidor

En esta ocasión veremos como desplegar los archivos que tenemos en el servidor en un TreePanel, luego mediante un panel desplegaremos el contenido del archivo cuando el usuario de click sobre este.

Ya vimos como leer el contenido de una carpeta en el servidor con PHP y también lo desplegamos en un TreePanel utilizando Ajax, en este tutorial haremos exactamente lo mismo pero le agregaremos la funcionalidad de permitir al usuario ver en un panel el contenido del archivo seleccionado, sea una imagen, un archivo de texto, un Html o inclusive un Pdf.

He preparado una demostración de lo que obtendremos al final de este tutorial, te recomiendo le des un vistazo para tener una idea clara de lo que estaremos haciendo.

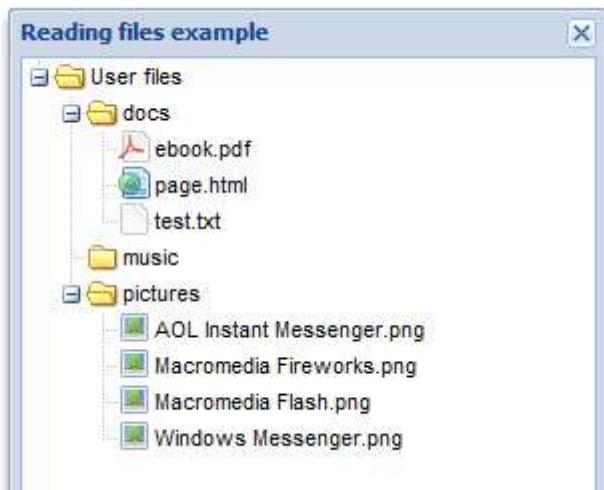


Demostración del tutorial

### Material de apoyo

El material de apoyo es el código que escribimos en el tutorial anterior, trabajaremos sobre lo hecho para evitar explicaciones, no modificaremos el archivo PHP que lee los archivos, solo modificaremos el JavaScript.

Una vez que copiemos todos los archivos que descargamos podremos ver algo como la siguiente imagen en nuestro explorador:



Utilizamos el código del tutorial anterior

### Creando el Layout

Primero vamos a crear el layout que tendrá nuestra aplicación, necesitamos que en la parte izquierda aparezca el TreePanel y en la parte derecha aparezca el panel donde cargaremos el contenido de cada archivo.

Vamos a modificar el código de la ventana de la siguiente manera.

```

1. var win = new Ext.Window({
2. title:'Reading files example',
3. layout:'border', // Step 1
4. width:600, // Step 2
5. height:400,
6. items: tree
7. });
8. win.show();

```

En el paso uno cambiamos el layout de “fit” a “border”, recordemos que con el valor “fit” el contenido de la ventana se ajustará a las dimensiones de la misma ventana, pero ahora si usamos el layout “border” podremos crear “regiones” dentro de la ventana, en tutoriales siguientes explicaré más a detalle esto, por ahora es suficiente.

En el paso dos cambiamos las dimensiones de la ventana, dando espacio para agregar el panel donde se desplegará el contenido de cada archivo.

### **Crear las regiones**

Cuando usamos un layout de tipo “border” es necesario definir las regiones que contendrá, es importante mencionar que debemos crear una región “center” obligatoriamente, de lo contrario aparecerá un error como el siguiente:

1. uncaught exception: No center region defined in BorderLayout ext-comp-1002

El layout de tipo “border” cuenta con cinco regiones, norte (north), sur (south), este (east), oeste (west) y centro (center), ya sabemos que la región central es obligatoria, para este ejemplo usaremos la región “oeste” para colocar el TreePanel y la región “central” para poner el contenido del archivo seleccionado.

```

1. var tree = new Ext.tree.TreePanel({
2. region:'west', // Step 1
3. border: false,
4. autoScroll:true,
5. dataUrl:'get-files.php',
6. root: new Ext.tree.AsyncTreeNode({
7. id:'',
8. text: 'User files'
9. })
10. });
11.
12. var main = new Ext.Panel({
13. region:'center', // Step 2
14. layout:'fit',
15. border:false
16. });
17.
18. var win = new Ext.Window({
19. title:'Reading files example',
20. layout:'border',
21. width:600,
22. height:400,
23. items: [tree,main] // Step 3
24. });

```

En el paso uno le asignamos al TreePanel la región “west”, con esto se alinearán en la parte izquierda de la ventana contenedora.

En el paso dos creamos el panel central, aquí no hay nada diferente solamente la región es lo que debemos poner atención.

En el paso tres le asignamos a la ventana los dos elementos que necesitamos que contenga, si creamos más regiones aquí las asignaríamos.



Layout border con dos regiones

#### Permitir que se redimensione el TreePanel

Vamos asignarle algunas otras configuraciones al TreePanel con el fin de que el usuario pueda cambiar su tamaño y que también lo pueda ocultar.

```
1. var tree = new Ext.tree.TreePanel({
2. region:'west',
3. split:true, // Step 1
4. collapsible:true, // Step 2
5. title:'Files', // Step 3
6. width:130,
7. border: false,
8. autoScroll:true,
9. dataUrl:'get-files.php',
10. root: new Ext.tree.AsyncTreeNode({
11. id:'.',
12. text: 'User files'
13. })
14.});
```

En el paso uno hicimos que el panel se pueda redimensionar, con esto permitimos que el usuario tenga la capacidad de cambiar el ancho del panel.

En el paso dos hacemos que el panel pueda colapsarse, esto permite que pueda ocultarse para tener más espacio y ver mejor el contenido principal.

En el paso tres le asignamos un título y también hemos cambiado la dimensión inicial.

Al actualizar el explorador podremos redimensionar el TreePanel y veremos algo como la siguiente imagen:



Redimensionar el TreePanel

Y si damos click sobre la flecha que se encuentra en el título del TreePanel, éste se colapsará de la siguiente manera:



Ocultar el TreePanel

### Desplegando el contenido de los archivos

Hasta ahora nuestra aplicación no hace nada, solo despliega los archivos que el servidor nos permite ver, necesitamos agregar más interacción para el usuario y esto lo logramos asignándole los eventos adecuados a los componentes.

```
1. tree.on('click',function(node){
2. // Solicitar el contenido del nodo seleccionado
3. });
```

En el código anterior estamos asignándole un listener al evento "click" del TreePanel, de esta manera cuando el usuario seleccione un archivo la función que definimos será ejecutada y recibirá como parámetro el "nodo" seleccionado.

Lo siguiente es desplegar el archivo seleccionado, para esto usaremos un "iframe" al cual le asignaremos la "url" que contiene el nodo.

```
1. // Step 1
2. var el = Ext.get(Ext.DomQuery.select('.x-panel-body',main.el.dom)[0]);
3.
4. el.createChild({ // Step 2
5. tag:'iframe',
6. src:'files/'+node.id,
7. style:'border:none;width:100%;height:100%;'
8. });
```

En el paso uno obtenemos el div del panel central, este div es necesario pues ahí es donde colocaremos el "iframe".

En el paso dos creamos un "iframe" dentro del div que tomamos anteriormente, este "iframe" que apunta hacia el documento seleccionado, también le agregamos algunos estilos.



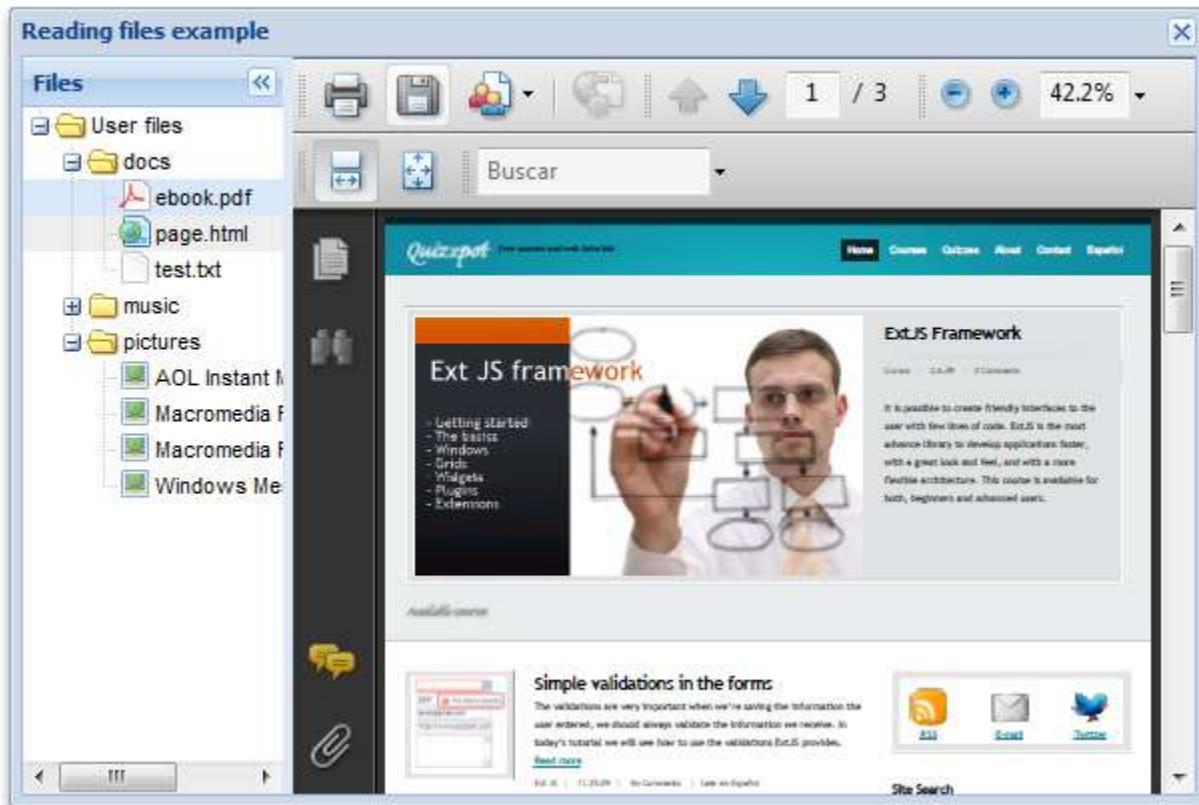
Mostrar el contenido de los archivos

Ya podemos desplegar el contenido de los archivos, pero tenemos un problema, una vez que seleccionamos un archivo al dar clic sobre otros ya no se despliegan en el panel principal, esto sucede porque los iFrames se van creando por debajo de los anteriores, para solucionar este problema solamente tenemos que eliminar los iFrames anteriores.

```
1. tree.on('click',function(node){
2. var el = Ext.get(Ext.DomQuery.select('.x-panel-body',main.el.dom)[0]);
3.
4. while (el.dom.childNodes[0]){ // Step 1
5. el.dom.removeChild(el.dom.childNodes[0]); //Step 2
6. }
7.
8. el.createChild({
9. tag:'iframe',
10. src:'files/'+node.id,
11. style:'border:none;width:100%;height:100%;'
12. });
13.});
```

En el paso uno creamos un ciclo que se estará ejecutando hasta que no haya más elementos dentro del div principal.

En el paso dos eliminamos el primer nodo del div principal.



Mostrar los archivos seleccionados

Ahora si funciona correctamente nuestra aplicación, podemos ver el contenido de los archivos que tenemos en el servidor.

### Conclusión

En este tema vimos como crear una layout de tipo “border”, no entramos en detalles sobre este tipo de layout ya que la idea es hacerlo más adelante, también vimos como utilizar el evento “click” en un TreePanel.

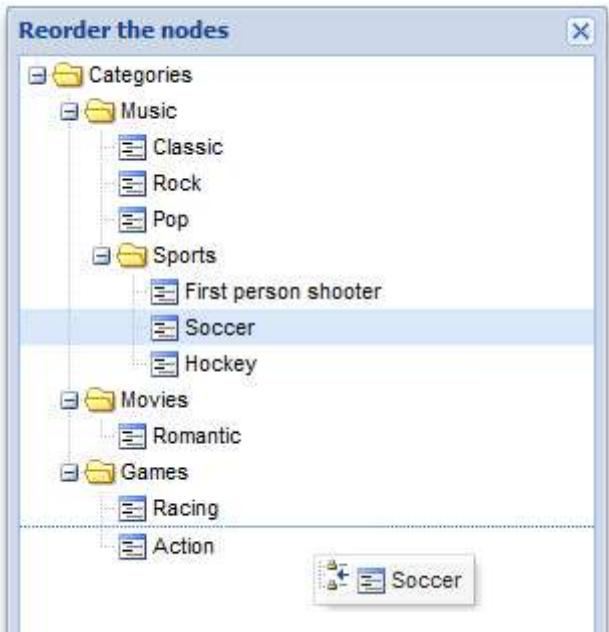
## Ordenar los nodos de un TreePanel

En el tutorial de hoy veremos cómo podemos ordenar los elementos del TreePanel y además los guardaremos en una base de datos MySQL, este es el primer tutorial donde mostraré un poco más allá del uso de ExtJS y hablare de cómo generar el árbol a partir de una tabla en la base de datos.

Supongamos que tenemos una tienda en línea, los productos que se venderán estarán categorizados para que el usuario final pueda navegar fácilmente por la tienda, una categoría puede estar contenida dentro de otra categoría principal, por ejemplo podríamos tener la categoría “Música” y tener alguna subcategoría como “Rock”, el número de niveles será definido por el administrador de la tienda, así que no pondremos límite de niveles.

### La demostración

El día de hoy solamente veremos cómo podemos ordenar las categorías existentes, en el siguiente tutorial prometo mostrar cómo podemos agregar o eliminar categorías. Te recomiendo visitar el ejemplo que he preparado.



Ejemplo del tutorial

### Material de apoyo

Antes de continuar descarguemos el material de apoyo para este tutorial. Lo descomprimimos y copiamos los archivos a nuestro servidor Web que instalamos al inicio del curso, esto es necesario porque usaremos Ajax, si es necesario cambia las rutas a la librería ExtJS dentro del archivo "tree-order.html".

### Creación de la base de datos

Lo primero que haremos será crear la tabla que usaremos para almacenar las categorías, vamos a crear una tabla que se llame "categories" y que contenga los campos "id, id\_parent, category, order\_number y description".

| Server: localhost > Database: testing > Table: categories |                     |              |                   |            |      |         |                |
|-----------------------------------------------------------|---------------------|--------------|-------------------|------------|------|---------|----------------|
|                                                           | Field               | Type         | Collation         | Attributes | Null | Default | Extra          |
| <input type="checkbox"/>                                  | <b>id</b>           | int(11)      |                   |            | No   | None    | auto_increment |
| <input type="checkbox"/>                                  | <b>id_parent</b>    | int(11)      |                   |            | Yes  | NULL    |                |
| <input type="checkbox"/>                                  | <b>category</b>     | varchar(255) | latin1_swedish_ci |            | No   | None    |                |
| <input type="checkbox"/>                                  | <b>order_number</b> | int(11)      |                   |            | Yes  | NULL    |                |
| <input type="checkbox"/>                                  | <b>description</b>  | text         | latin1_swedish_ci |            | No   | None    |                |

Check All / Uncheck All With selected:

Descripción de la tabla "categories"

Para los que solo les gusta copiar/pegar aquí les dejo el SQL necesario para hacer lo que describí anteriormente.

1. CREATE TABLE categories (
2.     id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY ,
3.     id\_parent INT NULL ,
4.     category VARCHAR( 255 ) NOT NULL ,
5.     order\_number INT NULL ,
6.     description TEXT NOT NULL
7. ) ENGINE = INNODB;

Ahora vamos a crear algunas categorías para que podemos probar el funcionamiento del tutorial, aquí dejo unos cuantos "inserts" a la tabla "categories":

1. INSERT INTO categories (id, id\_parent, category, description)
2. VALUES (NULL, NULL, 'Music', 'The music category'),
3. (NULL, NULL, 'Movies', 'All about movies and stuff'),
4. (NULL, NULL, 'Games', 'Are you a gamer?'),
5. (NULL, 1, 'Classic', 'For those who love a quiet moment'),
6. (NULL, 1, 'Rock', 'Let's rock and roll!'),
7. (NULL, 1, 'Pop', 'Do you like pop music?'),
8. (NULL, 2, 'Action', 'Actions movies for everyone'),
9. (NULL, 2, 'Romantic', 'Watch this with your girlfriend'),
10. (NULL, 3, 'Sports', 'Want to play a game?'),
11. (NULL, 3, 'First person shooter', 'Let's kill some fools'),
12. (NULL, 3, 'Racing', 'For those who like to drive!'),
13. (NULL, 9, 'Soccer', 'All about foot ball soccer'),
14. (NULL, 9, 'Hockey', 'Ready for the ice?');

Aquí lo único interesante es el campo "id\_parent", este campo tendrá el "id" de la categoría "padre", de esta manera podremos construir una estructura de árbol con "n" cantidad de niveles, los otros campos creo que son muy descriptivos por lo tanto no hablaré de ellos.

|                          | <b>id</b> | <b>id_parent</b> | <b>category</b> | <b>order_number</b> | <b>description</b>   |
|--------------------------|-----------|------------------|-----------------|---------------------|----------------------|
| <input type="checkbox"/> |           | X                | 1               | NULL                | Music                |
| <input type="checkbox"/> |           | X                | 2               | NULL                | Movies               |
| <input type="checkbox"/> |           | X                | 3               | NULL                | Games                |
| <input type="checkbox"/> |           | X                | 4               | 1                   | Clasic               |
| <input type="checkbox"/> |           | X                | 5               | 1                   | Rock                 |
| <input type="checkbox"/> |           | X                | 6               | 1                   | Pop                  |
| <input type="checkbox"/> |           | X                | 7               | 2                   | Action               |
| <input type="checkbox"/> |           | X                | 8               | 2                   | Romantic             |
| <input type="checkbox"/> |           | X                | 9               | 3                   | Sports               |
| <input type="checkbox"/> |           | X                | 10              | 3                   | First person shooter |
| <input type="checkbox"/> |           | X                | 11              | 3                   | Racing               |
| <input type="checkbox"/> |           | X                | 12              | 9                   | Soccer               |
| <input type="checkbox"/> |           | X                | 13              | 9                   | Hockey               |

Check All / Uncheck All With selected:

Información de prueba

### Creación de la estructura de árbol

El siguiente paso es crear una estructura de árbol con los registros que tenemos en nuestra base de datos, he creado una clase en PHP que nos hará la vida mucho más sencilla.

```

1. /**
2. * This class creates a Tree structure of information for the TreePanel component
3. * of the ExtJS library.
4. *
5. * @author Crysfel Villa
6. * @date 12/18/2009
7. *
8. */

```

```

9. class TreeExtJS{
10. private $tree = array();
11. private $index = array();
12. private $cont = 0;
13.
14. /**
15. * This method inserts a node to the Tree, the child param may contain an
16. * "id" property that will be used as a "key", if the child param doesn't contain
17. * an "id" property a generated "key" is given to the node.
18. *
19. * @child the node to insert
20. * @parentKey(optional) The parent key where the node will be inserted, if null
21. * the node is inserted in the root of the Tree
22. */
23. public function addChild($child,$parentKey = null){
24. $key = isset($child["id"])?$child["id"]:'item_'.$this->cont;
25. $child["leaf"] = true;
26. if($this->containsKey($parentKey)){
27. //added to the existing node
28. $this->index[$key] =& $child;
29. $parent =& $this->index[$parentKey];
30. if(isset($parent["children"])){
31. $parent["children"][] =& $child;
32. }else{
33. $parent["leaf"] = false;
34. $parent["children"] = array();
35. $parent["children"][] =& $child;
36. }
37. }else{
38. //added to the root
39. $this->index[$key] =& $child;
40. $this->tree[] =& $child;
41. }
42. $this->cont++;
43. }
44.
45. /**
46. * Return a node by the given key
47. * @key
48. */
49. public function getNode($key){
50. return $this->index[$key];
51. }
52.
53. /**
54. * @TODO Remove the node from the Tree
55. * @key
56. */
57. public function removeNode($key){
58. //unset($this->index[$key]);
59. }
60.
61. /**
62. * Check if exist a node with the given key
63. */
64. public function containsKey($key){
65. return isset($this->index[$key]);
66. }
67.
68. /**

```

```

69. * Return a representation of the Tree structure in JSON format
70. */
71. public function toJson(){
72. return json_encode($this->tree);
73. }
74. }

```

He documentado los métodos que contiene esta clase, te recomiendo copiar/pegar el código en un archivo que se llame “TreeExtJs.class.php” (ya viene en el material de apoyo), la idea de esta clase es ir insertando nodos dentro de algún nodo existente en el árbol o bien a la raíz, una vez construida la estructura podremos generar una representación de esta en formato JSON para que se lo envíemos al componente TreePanel y este pueda desplegarla correctamente.

Es importante mencionar que es la primera versión de esta clase, y que en el futuro planeo agregarle más funcionalidades y mejoras, pero por ahora hace lo necesario para este tutorial.

### Generar el JSON a desplegar

Ahora tenemos que conectarnos a la base de datos y sacar la información que necesitamos desplegar, vamos a escribir el siguiente código dentro el archivo “tree-order.php”:

```

1. // Include the Tree class to generate the TreePanel JSON easily
2. include("TreeExtJS.class.php");
3.
4. // Make a MySQL Connection and select the database
5. $conn = mysql_connect("localhost", "root", "") or die(mysql_error());
6. mysql_select_db("testing") or die(mysql_error());
7.
8. // Retrieve all the data from the "categories" table
9. $result = mysql_query("SELECT * FROM categories order by id_parent,order_number asc") or di
e(mysql_error());

```

Notemos que estamos haciendo uso de “order by” para que las categorías sean regresadas ordenadas por el “id\_parent” y “order\_number”, esto es de suma importancia ya que de esta manera nos será más sencillo crear la estructura de árbol.

Lo siguiente es crear un “array” con la información que hemos sacado de la base de datos, esto lo hacemos de la siguiente manera:

```

1. // Create an array of the data
2. $data = array();
3. while($row = mysql_fetch_array($result)){
4. array_push($data,array(
5. "id" => $row["id"],
6. "idParent" => $row["id_parent"],
7. "text" => $row["category"],
8. "orderNumber" => $row["order_number"],
9. "description" => $row["description"]
10.));
11. }

```

Aquí no hay nada complicado, solamente estamos “populando” la colección “\$data” con los registros de la base de datos.

Por último necesitamos crear la estructura de árbol haciendo uso de la clase “TreeExtJs”, esto será muy sencillo.

```

1. // Creating the Tree
2. $tree = new TreeExtJS();
3. for($i=0;$i<count($data);$i++){
4. $category = $data[$i];

```

```

5. $tree->addChild($category,$category["idParent"]);
6. }
7.
8. echo $tree->toJson();

```

Mediante un ciclo vamos recorriendo la información que sacamos de la base de datos y vamos agregando nuevos nodos al árbol, aquí es donde es de suma importancia que vengan ordenados los elementos por el campo “id\_parent” ya debemos insertar primero los nodos padre para que al insertar los hijos no se ocasione ningún error.

No olvidemos cerrar la conexión a la base de datos.

```
1. mysql_close($conn);
```

Hemos terminado la parte más complicada de este tutorial, ahora vamos a escribir el JavaScript necesario para desplegar correctamente la información que creamos en el servidor.

### Creación del TreePanel

Editamos el archivo “tree-order.js” y dentro de la función “init” creamos el componente que necesitamos.

```

1. var tree = new Ext.tree.TreePanel({
2. border: false,
3. autoScroll:true,
4. dataUrl:'tree-order.php',
5. enableDD: true, // Step 1
6. root: new Ext.tree.AsyncTreeNode({
7. text: 'Categories',
8. draggable: false //Step 2
9. })
10. });
11.
12. var win = new Ext.Window({
13. title:'Reorder the nodes',
14. layout:'fit',
15. width:300,
16. height:400,
17. items: tree
18. });
19. win.show();

```

Anteriormente hemos visto como crear un TreePanel, en el código anterior solamente hay dos diferencias con respecto a los tutoriales pasados.

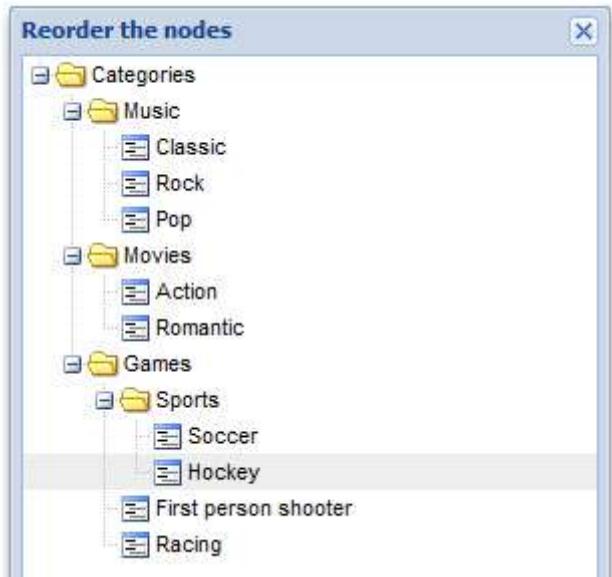
En el paso uno habilitamos que los nodos puedan ser arrastrados y soltados (Drag and Drop), esto nos permitirá que el usuario pueda reacomodar las categorías a su gusto.

En el paso dos le indicamos al nodo “root” que no se pueda arrastrar, esto es importante para que el usuario no pueda mover el nodo principal.



Creación de un TreePanel

Si colapsamos el nodo principal podremos ver la información que tenemos en nuestra base de datos.



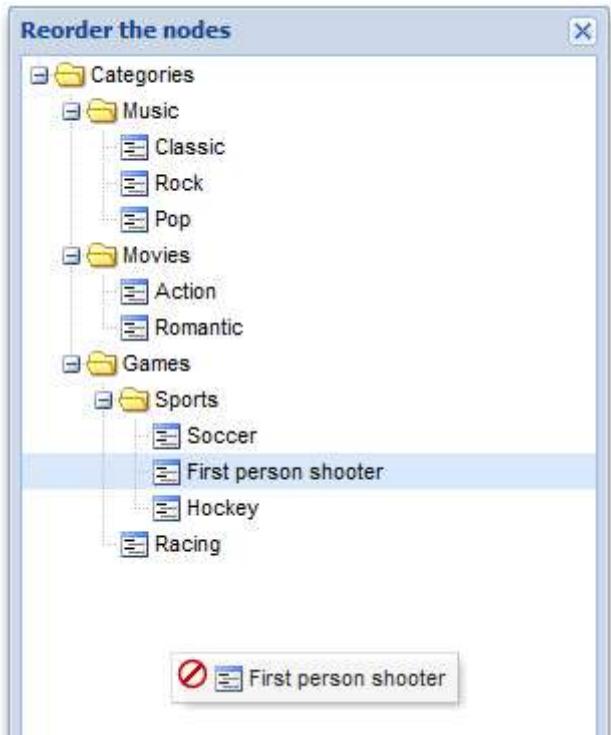
Información cargada de la base de datos MySQL

También podemos arrastrar los nodos de un lugar a otro con el mouse.



Drag and Drop a los elementos

El mismo componente se encarga de monitorear donde se soltará el nodo que se está arrastrando. Si es un lugar incorrecto simplemente no permitirá que sea soltado avisando al usuario con un ícono en color rojo.



Restricciones del componente

### Guardando el orden

Hasta ahora podemos ordenar los elementos en el TreePanel, pero si actualizamos la página vemos que el orden es el mismo que teníamos al principio, para que los cambios puedan persistir debemos guardar los cambios efectuados en nuestra base de datos.

Debemos buscar el evento adecuado para guardar los cambios, podríamos crear un botón “guardar” para que el usuario guarde los cambios cuando crea que es necesario, o bien podríamos hacerlo más transparente al usuario y guardar los cambios automáticamente cuando suceda cualquier movimiento en los nodos.

Para este tutorial haremos la segunda opción, para eso debemos utilizar el evento “movenode” que se disparará cuando un nodo cambió de lugar.

```
1. tree.on('movenode',function(tree,node,oldParent,newParent,index){
2. //save the new order
3. });


```

Podemos ver en los parámetros que uno lleva por nombre “newParent”, necesitamos utilizar este parámetro para iterar por todos sus hijos de primera generación, es decir aquellos que sean descendientes directos de él, luego sacamos el “ID” de cada uno de ellos para enviarlos en el orden correcto al servidor y que éste los guarde en la base de datos.

```
1. var nodes = [];
2. newParent.eachChild(function(n){
3. nodes.push(n.attributes.id); //Step 1
4. });
5. tree.el.mask('Saving...', 'x-mask-loading'); //Step 2
6. Ext.Ajax.request({
7. url: 'tree-order.php', //Step 3
8. params:{ //Step 4
9. updateOrder: true,
10. parent: newParent.attributes.id,
11. nodes: nodes.join(',')}


```

```

12. },
13. success: function(){
14. tree.el.unmask(); //Step 5
15. },
16. failure: function(){
17. tree.el.unmask(); //Step 6
18. Ext.Msg.alert('Error','Error saving the changes');
19. }
20. });

```

En el paso uno recolectamos los “ID” de cada nodo, esto es necesario para generar el orden que guardaremos en la base de datos.

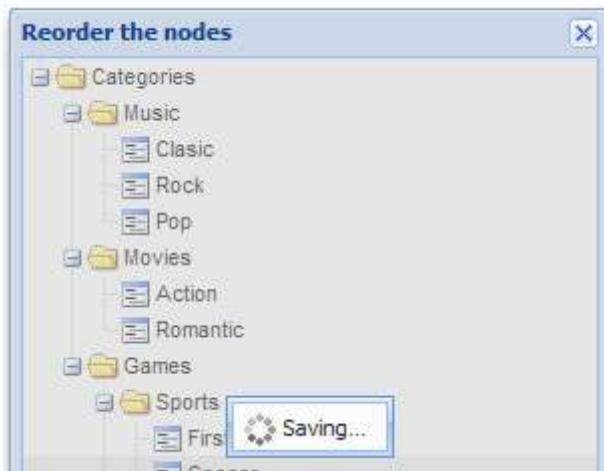
En el paso dos enmascaramos el TreePanel y mostramos un mensaje de “Saving...” esto es importante para que el usuario se dé cuenta que se están guardando sus cambios.

En el paso tres indicamos la “url” que procesará la petición Ajax, en este caso usaremos el mismo PHP, pero si estuviésemos usando algún Framework como: CodeIgniter, Ruby on Rails, Groovy and Grails, etc., pondríamos un “controller” diferente.

En el paso cuatro definimos los parámetros que enviaremos al servidor cuando se realice la petición Ajax, estos parámetros son “updateOrder” con el cual decidiremos si vamos hacer un “update” o un “select” a la tabla en la base de datos, el parámetro “parent” es importante porque permitiremos que el usuario cambie los nodos inclusive a otras “ramas”, por lo tanto necesitamos saber el padre de los nodos que enviamos en el parámetro “nodos”.

El paso cinco se ejecutará si se realizó satisfactoriamente la actualización, aquí solamente quitamos la máscara que pusimos al inicio de la petición Ajax.

El paso seis se ejecutará si ha sucedido algún error en el servidor o en la petición, aquí solamente quitamos la máscara y enviamos un mensaje de error al usuario para informarle que los cambios no se guardaron.



Guardando los cambios den la base de datos

Por el momento es todo lo que escribiremos en JavaScript ahora es el momento de hacer la lógica necesaria en el servidor para que guardemos los cambios efectuados.

En el archivo “tree-order.php” escribiremos el siguiente código después de abrir la conexión a la base de datos:

1. // Include the Tree class to generate the TreePanel JSON easily
2. include("TreeExtJS.class.php");
- 3.

```

4. // Make a MySQL Connection and select the database
5. $conn = mysql_connect("localhost", "root", "") or die(mysql_error());
6. mysql_select_db("testing") or die(mysql_error());
7.
8. if(isset($_POST["updateOrder"])){ // Step 1
9. //Aquí vamos a crear el UPDATE a la tabla categories
10. }else{
11. //aquí dejamos el código necesario para generar el JSON del...
12. // TreePanel que escribimos anteriormente
13. }
14.
15. mysql_close($conn);

```

En el paso uno solamente estamos tomando la decisión de hacer el UPDATE o hacer el SELECT, esto es definido por el parámetro “updateOrder”, si el parámetro existe en el request haremos un UPDATE a la tabla, si no existe entonces se hace todo lo que ya escribimos anteriormente.

Para guardar los cambios lo hacemos de la siguiente manera:

```

1. $nodes = $_POST["nodes"]; //Step 1
2. $ids = explode(",",$nodes);
3. $idParent = (int)$_POST["parent"]; //Step 2
4. for($i=0;$i<count($ids);$i++){
5. $id = (int)$ids[$i];
6. $query = sprintf("UPDATE categories SET order_number = %d,id_parent = %d WHERE id = %d",
7. $i,$idParent,$id); //Step 3
8. mysql_query($query); //Step 4
9. }

```

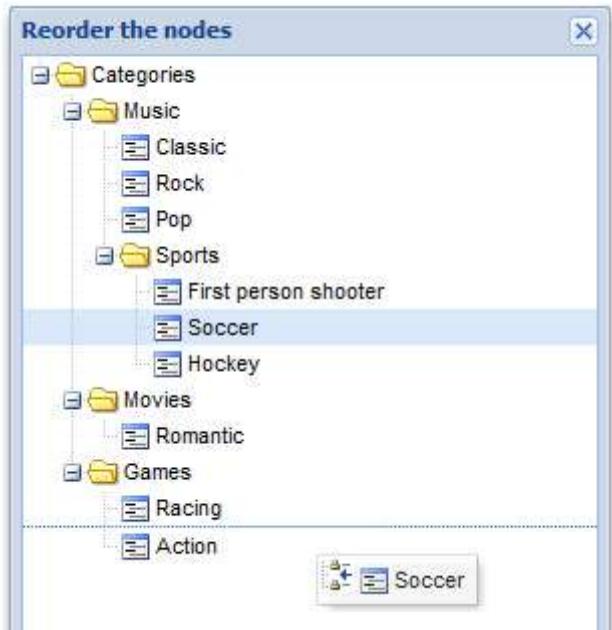
Primero tomamos los nodos que editaremos y creamos un arreglo con cada “ID”.

El segundo paso es tomar el parámetro “parent” y convertirlo a entero.

En el tercer paso creamos el query para actualizar el campo “order\_number” y “id\_parent”, lo hacemos de esta manera para asegurarnos que no hagan un SQL Injection, los parámetros “id” e “idParent” deben ser enteros.

En el paso cuatro ejecutamos el query para que se guarden los cambios correspondientes.

Ahora solo queda probar la aplicación, debería ser capaz de mostrar y guardar los cambios efectuados inclusive si la ejecutamos de diferentes computadoras, esto es posible porque la información esta contenida en la base de datos y no solamente en la memoria de nuestra computadora u ordenador.



El orden persiste

### Conclusiones

Hemos visto como es muy sencillo ordenar los nodos de un TreePanel, la parte más complicada en mi opinión es crear la estructura de arbol, esto lo había comentado en tutoriales anteriores pero ahora hemos visto como crearla.

## Layouts y maquetación

Los layout se utilizan para crear la maquetación de un sitio, ExtJS provee maneras diferentes para lograr este objetivo, en este capítulo se analizan algunas.

### Border layout

ExtJS cuenta con una excelente manera de construir nuestras interfaces, el día de hoy estudiaremos el layout de tipo "border" el cual nos permite dividir un panel o contenedor en regiones.

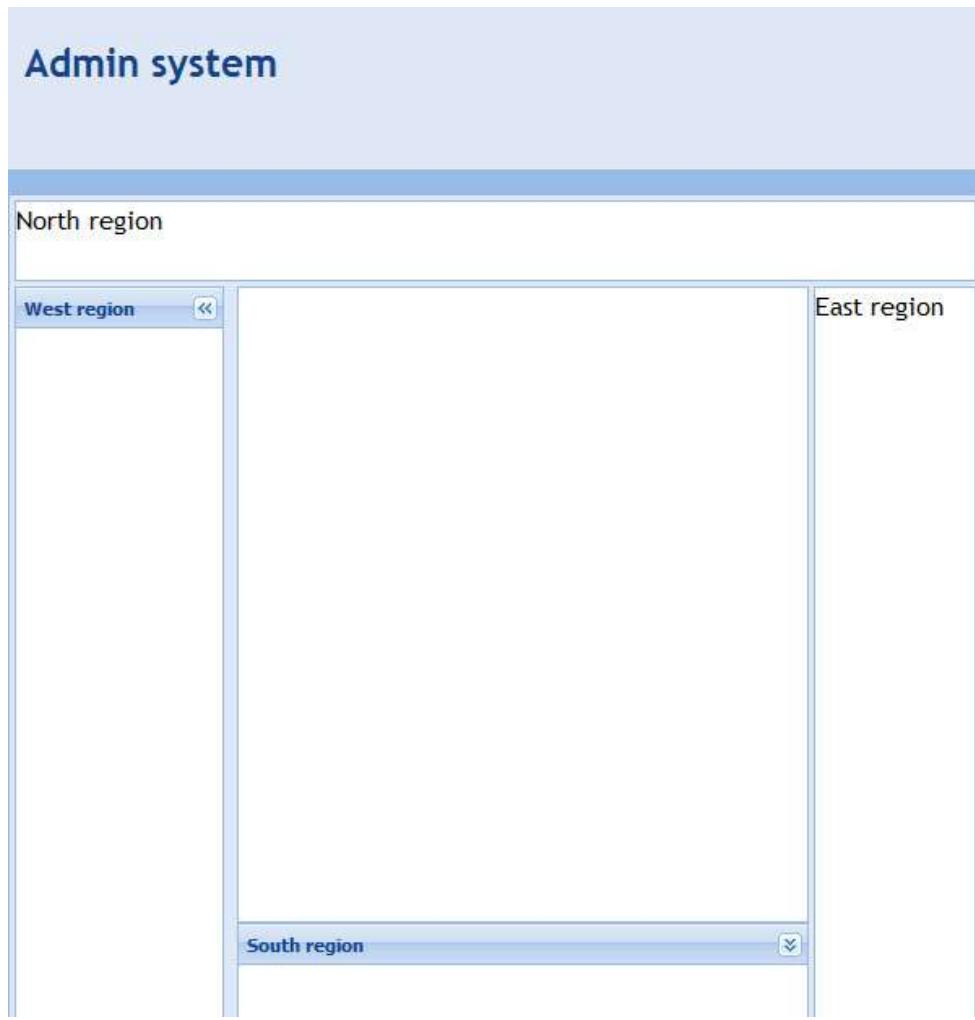
xisten diferentes tipos de layouts por ejemplo: accordion, border, absolute, column, fit, form, table, etc. Cada uno nos permite acomodar nuestros bloques de maneras diferentes, además de que nos proporcionan algunos funcionamientos predeterminados.

#### Material de apoyo

Vamos a descargar el material de apoyo, lo descomprimes y copias dentro del servidor Web que instalamos al inicio del curso, actualiza las rutas a la librería de ExtJS en el HTML, para este tutorial usaremos la versión 3.1.

Si lo deseas puedes descargar el código fuente, podría servirte como referencia.

También puedes ver la demostración de lo que tendremos al final del tutorial.



### Definición de un layout de tipo border

Si utilizamos un layout de tipo “border” podremos crear cuatro regiones norte (north), sur (south), este (east) y oeste (west), dentro de cada región podremos poner el contenido que necesitemos, es importante mencionar que también debemos crear una región central (center) en donde colocaremos el contenido principal.

Lo primero que haremos es crear un panel y renderizarlo en el div con identificador “content” de la siguiente manera:

```
1. Ext.ns("com.quizzpot.tutorial");
2.
3. com.quizzpot.tutorial.BorderLayoutTutorial = {
4. init: function(){
5.
6. //code goes here
7. var main = new Ext.Panel({ //Step 1
8. renderTo : "content",
9. layout : "border", // Step 2
10. height : 600,
11. html : "Hey everyone!"
12. });
13.
14. }
15. }
16.
17. Ext.onReady(com.quizzpot.tutorial.BorderLayoutTutorial.init,com.quizzpot.tutorial.BorderLayoutTutorial);
```

En el paso uno solamente creamos una instancia del componente Panel.

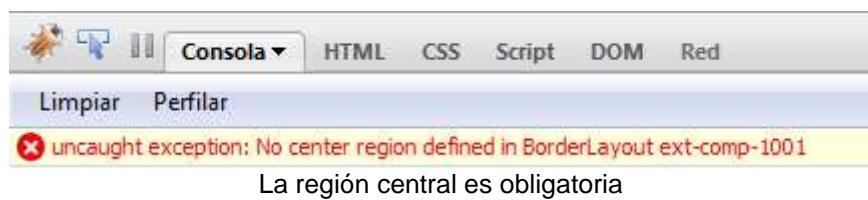
En el paso dos usamos la propiedad “layout” y le asignamos “border”, de esta manera podremos usar las regiones que mencioné anteriormente.



Crear el panel contenedor

Si en este momento actualizamos el navegador veremos que en la consola de Firebug nos aparecerá un error, el error dice de la siguiente manera:

“uncaught exception: No center region defined in BorderLayout ext-comp-1001”



Este error se ocasiona porque es obligatorio definir la región central, para solucionarlo solamente es necesario crear el panel principal y definir su región en "center".

```
1. var center = {
2. xtype : "panel",
3. region : "center",
4. html : "Center region"
5. };
6.
7. var main = new Ext.Panel({
8. renderTo : "content",
9. layout : "border",
10. height : 600,
11. items : [center]
12.});
```



Agregamos una región central al pánel principal

Con esto hemos solucionado el error, ahora vamos a crear un panel y lo posicionaremos en la región "west".

```
1. var west = {
2. xtype : "panel",
3. region : "west", //Step 1
4. width : 150, //Step 2
5. html : "West region"
6. };
7.
8. var main = new Ext.Panel({
9. renderTo : "content",
10. layout : "border",
11. height : 600,
12. items : [center,west] //Step 3
13.});
```

En el paso uno solamente le asignamos la región oeste (west) al panel, esto es suficiente para que se acomode en la parte izquierda del panel principal.

En el paso dos le asignamos el "width", esto es muy importante para que el panel tenga un tamaño fijo. En el paso tres solamente le agregamos el panel "west" al panel principal para que podamos verlo renderizado en la pantalla.

## Admin system

West region      Center region

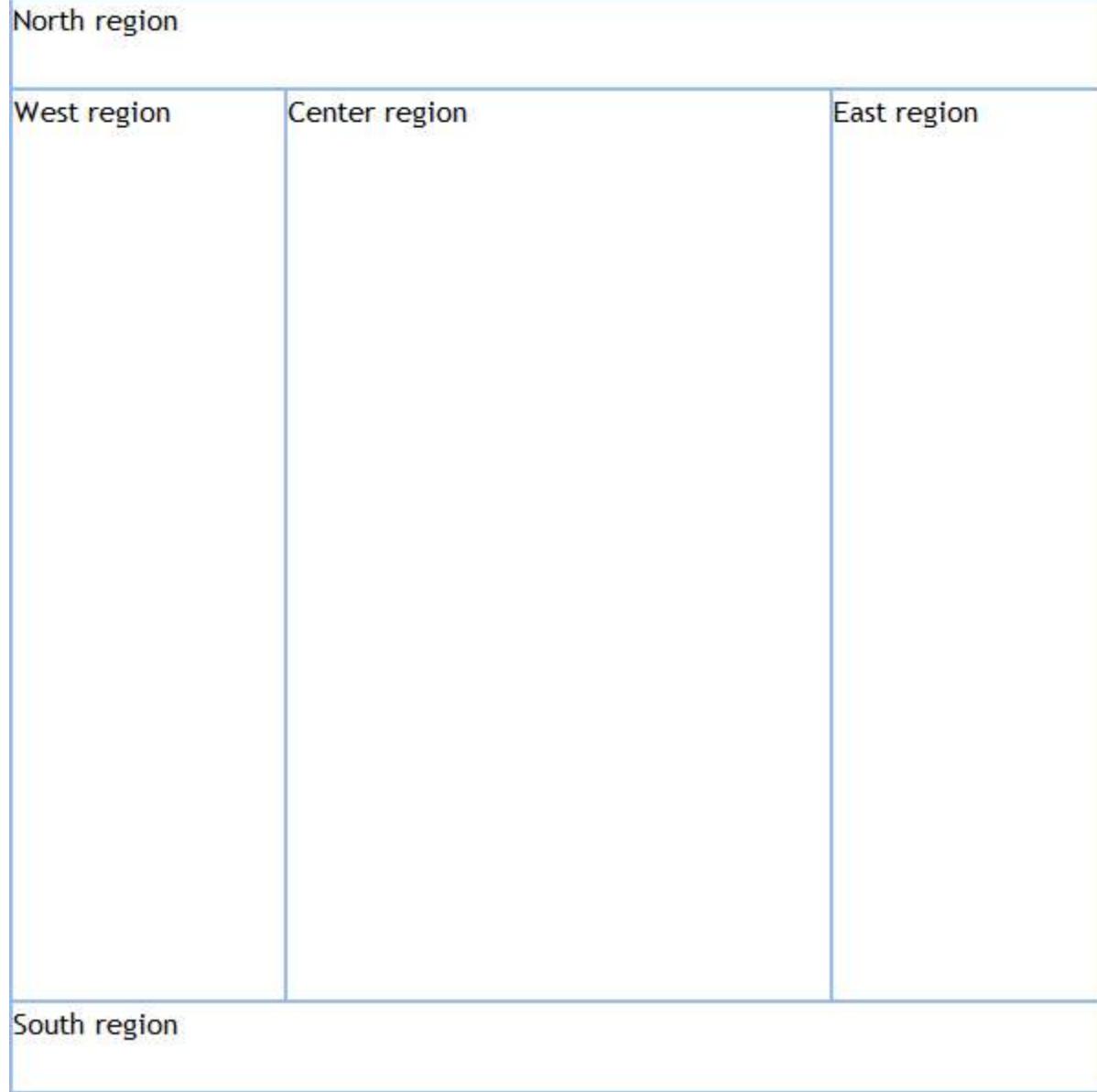
Región oeste

Para crear las otras regiones es exactamente igual a los pasos anteriores, solamente necesitamos indicarle la región deseada.

```
1. var east = {
2. xtype : "panel",
3. region : "east",
4. width : 150,
5. html : "East region"
6. };
7.
8. var north = {
9. xtype : "panel",
10. region : "north",
11. height : 50,
12. html : "North region"
13. };
14.
15. var south = {
16. xtype : "panel",
17. region : "south",
18. height : 50,
19. html : "South region"
20. };
21.
22. var main = new Ext.Panel({
23. renderTo : "content",
24. layout : "border",
25. height : 600,
26. items : [center,west,east,north,south]
27. });
```

En el código anterior solamente creamos los paneles asignándoles una región y luego los agregamos al panel principal.

# Admin system



Utilizando todas las regiones/p>

## Crear márgenes entre las regiones

Hasta ahora las regiones se dividen por un pequeño borde, podemos agregar algunos márgenes para que la diferencia se vea mejor, vamos a utilizar la propiedad "margins" la cual recibe un objeto con los márgenes que usaremos o también recibe un String con el valor de los márgenes en pixeles.

```
1. var north = {
2. xtype : "panel",
3. region : "north",
4. height : 50,
5. html : "North region",
6. margins: {top:3,bottom:3,left:3,right:3} //Step 1
```

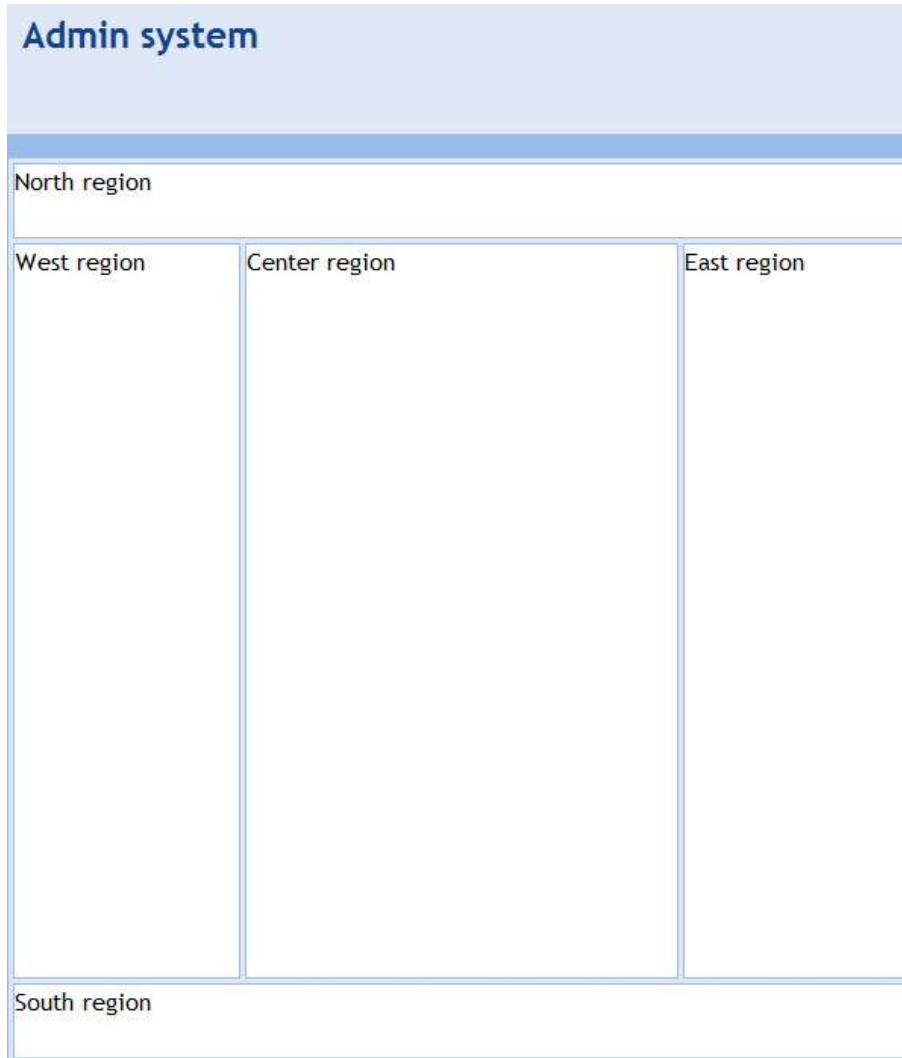
```

7. };
8.
9. var south = {
10. xtype : "panel",
11. region : "south",
12. height : 50,
13. margins: "3 3 3 3", //step 2
14. html : "South region"
15. };

```

En el paso uno le pasamos un objeto con los márgenes que tendrá el panel.

En el paso dos le asignamos un String con cada uno de los márgenes que usaremos.



Asignando márgenes a las regiones

### Redimensionando las regiones

También podemos permitir al usuario redimensionar las regiones con el mouse, para hacerlo solamente tenemos que agregar la propiedad "split" en el panel deseado.

```

1. var west = {
2. xtype : "panel",
3. region : "west",
4. width : 150,
5. split : true, //Step 1

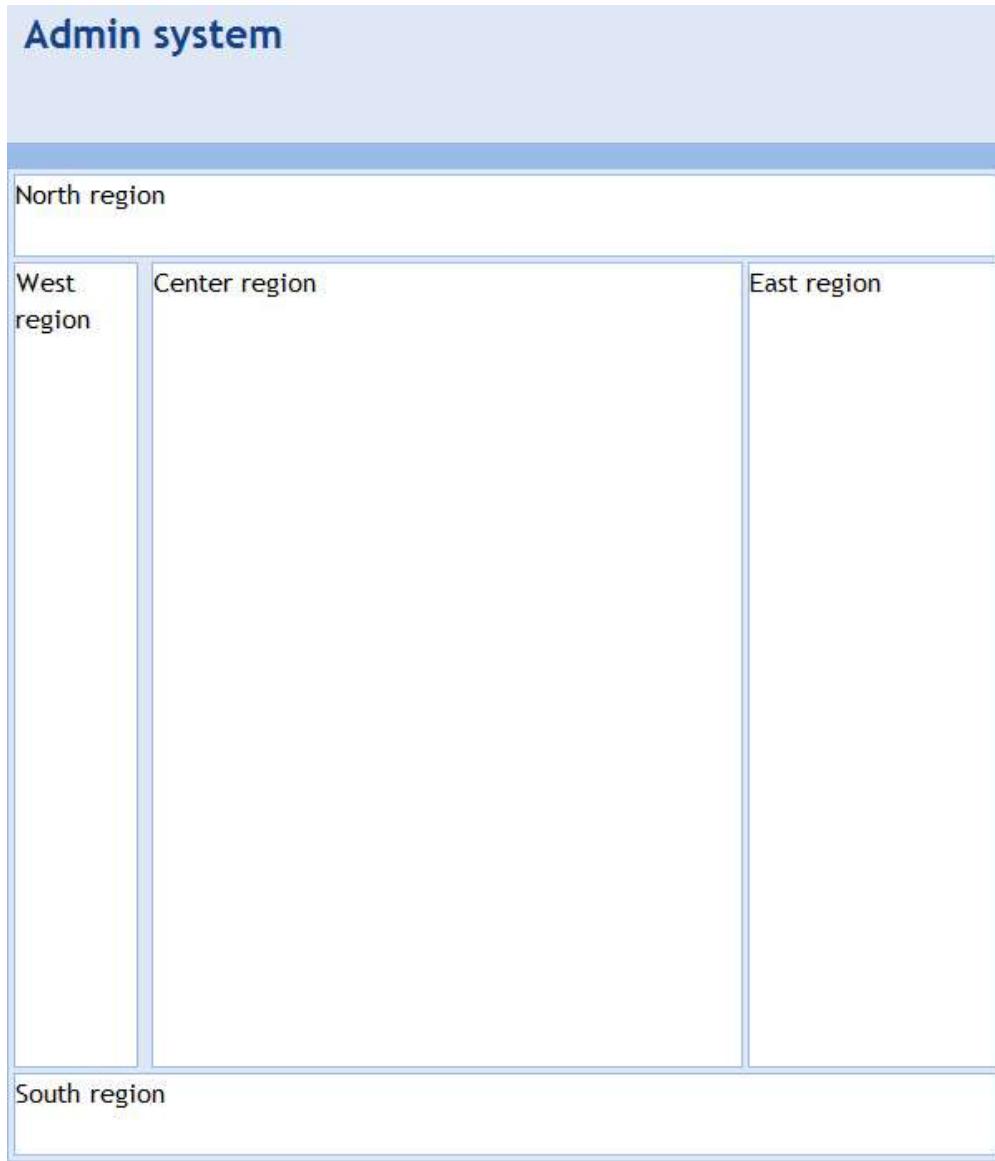
```

```

6. html : "West region",
7. margins: "0 3 0 3"
8. };

```

En el paso uno solamente le agregamos la propiedad “split” igual a “true” y automáticamente el panel podrá ser ajustado por el usuario.



La región del lado oeste puede ser redimensionada por el usuario

También podríamos hacer que la región se colapse, de esta manera podríamos permitir que el usuario oculte las regiones.

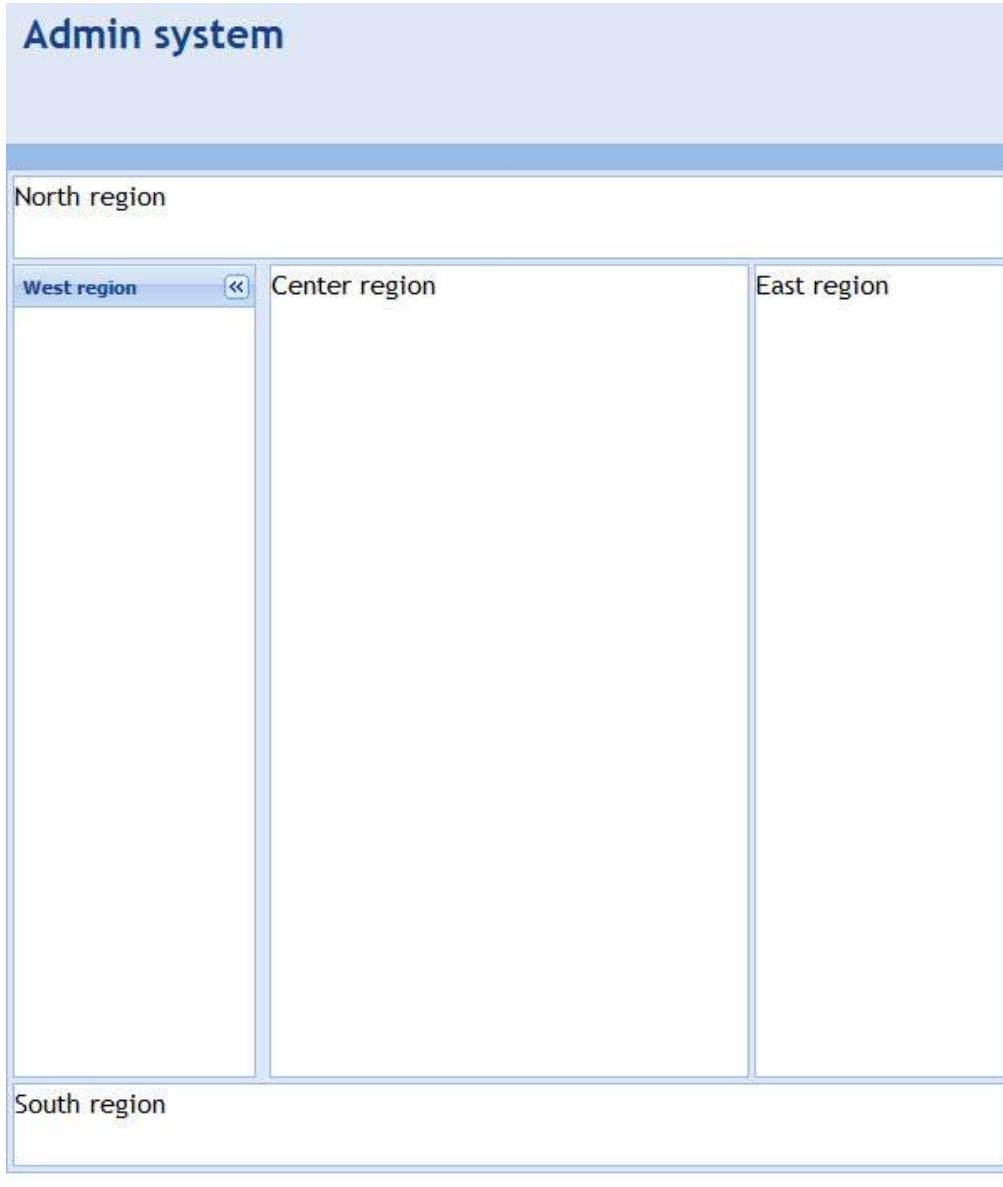
```

1. var west = {
2. xtype : "panel",
3. region : "west",
4. width : 150,
5. split : true,
6. collapsible: true, //Step 1
7. title : "West region", //Step 2
8. margins: "0 3 0 3"
9. };

```

En el paso uno usamos la propiedad “collapsible” para permitir que el panel se colapse al dar clic sobre el botón que aparece en el título.

En el paso dos solamente eliminamos la propiedad “html” que había anteriormente y asignamos la propiedad “title” para que el panel contenga un título.



La región oeste puede ocultarse o colapsarse

Es importante mencionar que podemos utilizar todas las propiedades de configuración que usamos en un panel, ya he hablado de este componente en tutoriales anteriores.

#### **Border Layout dentro de una región**

También podemos asignarle un layout “border” a una de las regiones que ya hemos definido, de esta manera podemos combinar varios tipos de layouts.

A continuación voy a eliminar la región del sur que tenemos hasta ahora y voy a asignarle a la región central otro layout de tipo “border” al cual también le asignaré una región central y una región al sur.

1. var south = {
2.     xtype : "panel",

```

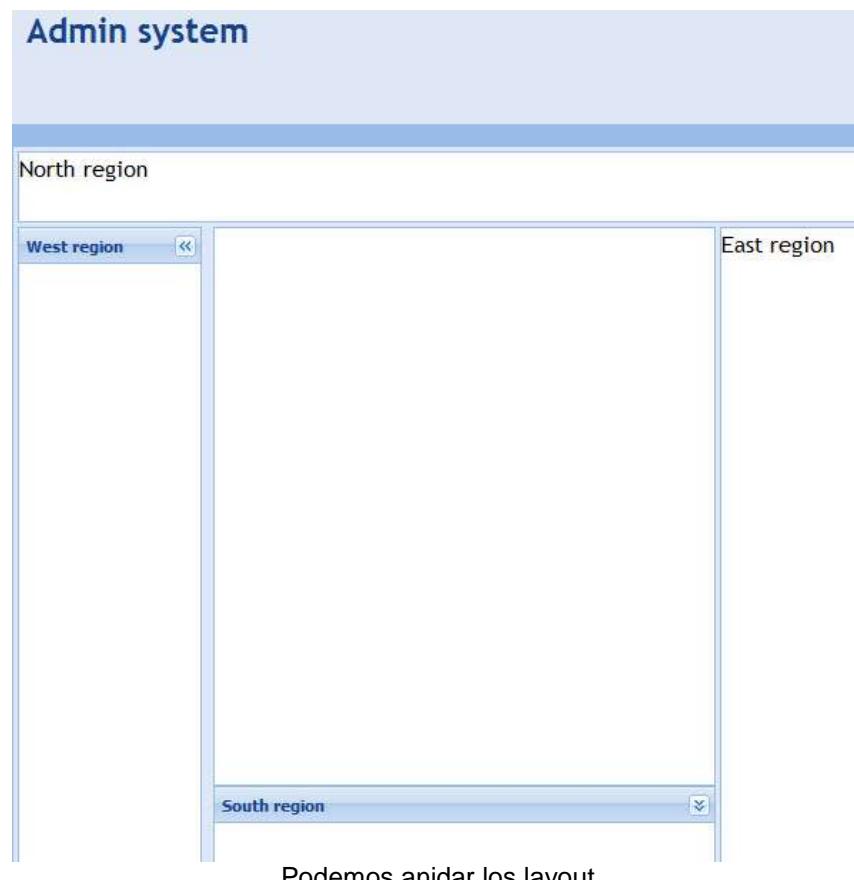
3. region : "south",
4. height : 150,
5. collapsible: true,
6. title : "South region"
7. };
8.
9. var center = {
10. xtype : "panel",
11. region : "center",
12. layout : "border", //Step 1
13. border : false,
14. items : [{
15. xtype : "panel",
16. region : "center" //Step 2
17. },
18. south //Step 3
19.]
20. };

```

En el paso uno le asignamos un layout de tipo border a la región central que ya teníamos.

En el paso dos le creamos la región central.

En el paso tres le asignamos la región sur.



Podemos anidar los layout

### Conclusiones

En el tutorial de hoy vimos como crear las regiones dentro de un panel con layout de tipo "border", este tipo de layout es muy usado en muchas aplicaciones ya que nos permite redimensionar o colapsar las regiones de una manera muy sencilla.

## Plugins y extensiones

La comunidad de ExtJS es muy activa, prueba de ello es la cantidad de plugins que se han desarrollado y son ofrecidos de manera gratuita, aquí se muestran algunos de ellos y como integrarlos a tu proyecto.

## Herencia con ExtJS

Ya hablamos un poco acerca de la herencia con JavaScript, vimos como hacerla utilizando los prototipos, el día de hoy quiero mostrar una manera muy elegante utilizando ExtJS para crear nuestras propias "clases" (prototipos, en JS no existen las clases) , componentes o extender de algún Panel, Ventana o lo que necesitemos.

Es necesario que comprendamos que en JavaScript no existe como tal el concepto de "clase", en JS utilizamos "prototipos" de los cuales generamos instancias, también podemos extenderlos utilizando la "herencia de prototipo", la manera de hacerlo es muy sencilla pero es un poco laboriosa. ExtJS nos provee de una utilería que nos hace la vida muchas más sencilla, mediante el método "Ext.extend" podemos crear "clases" (prototipos) de una manera muy elegante y luego podemos extenderlos para crear otras "clases" (prototipos) y reutilizar nuestro código las veces que necesitemos.

Podemos crear un prototipo de dos formas principales, y las analizaremos por separado.

### Crear prototipos a la manera antigua

Este método era propio de la versión 1.0 de ExtJS, esta era la manera de crear nuestros prototipos (clases), primero crear la función constructora y luego utilizar Ext.extend para terminar de crear el prototipo.

```
1. Ext.ns("com.quizzpot.tutorial"); // Step 1
2.
3. com.quizzpot.tutorial.Person = function(options){ //Step 2
4. Ext.apply(this,options || {}); //Step 3
5.
6. console.debug("Person constructor!");
7.
8. };
```

En el paso uno creamos el namespace que usará nuestro prototipo (clase), ya vimos las ventajas de hacerlo y porqué es necesario.

En el paso dos creamos la función constructora, esta función se ejecutará cada que se cree una instancia de nuestro prototipo, aquí podemos hacer todo lo que necesitemos para inicializar el nuevo objeto.

En el paso tres solamente estamos aplicando las opciones que el usuario definió para la nueva instancia, la idea de esto es que podamos cambiar las configuraciones por defecto de cada objeto.

Una vez que tenemos definido nuestro constructor podemos asignarle las propiedades y métodos que tendrá nuestro prototipo.

```
1. com.quizzpot.tutorial.Person = Ext.extend(com.quizzpot.tutorial.Person,{ //Step 1
2. username : "none", //Step 2
3. email : "none",
4. name : "none",
5. startTime : new Date(),
6. endTime : new Date(),
7.
8. checkIn : function(){ //Step 3
9. this.startTime = new Date();
10. },
11.
12. checkOut : function(){
13. this.endTime = new Date();
14. },
15.
```

```

15.
16. workingHours: function(){
17. return (this.endTime - this.startTime)/1000;
18. }
19.
20. });

```

En el paso uno hacemos uso del método “Ext.extend”, este método toma como primer argumento el objeto al cual queremos extender, en este caso a la función constructora solamente, pero podría ser cualquier otro componente que deseemos extender, el segundo parámetro es un objeto con el código que deseamos agregar o sobre escribir, en otras palabras la “subclase” (*?subprototipo? jobjeto!*), el método Ext.extend regresa un nuevo prototipo que contiene los método de la superclase (el prototipo original) y los métodos y propiedades del objeto nuevo.

En el paso dos solamente creamos unas propiedades, estas serán agregadas al prototipo de nuestro nuevo objeto.

En el paso tres creamos unos métodos.

### **Crear instancias del prototipo**

Una vez que tenemos nuestro prototipo listo podemos crear instancias de él muy fácilmente:

```

1. var crysfel = new com.quizzpot.tutorial.Person({ //Step 1
2. username : "crysfel",
3. email : "crysfel@dominio.com<script type="text/javascript">
4. /* <![CDATA[*/
5. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
=parseInt(a.substr(0,2),16);for(j=2;a.length-
j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s
;l.parentNode.replaceChild(s,l));}catch(e){}})();
6. /*]]> */
7. </script>",
8. name : "Crysfel Villa"
9. });
10.
11. crysfel.checkIn(); //Step 2
12. setTimeout(function(){
13. crysfel.checkOut();
14. var time = crysfel.workingHours();
15. console.debug(time+" seconds working!"); //Step 3
16. },3000);

```

En el paso uno creamos una instancia del prototipo utilizando el operador “new”, podríamos crear todas las que necesitemos cambiando sus configuraciones.

En el paso dos ejecutamos uno de los métodos de la instancia.

En el paso tres se imprime la cantidad de segundos transcurridos entre la hora de entrada y la hora de salida.

Como puedes ver es muy sencillo crear prototipos utilizando Ext.extend, ahora vamos a ver otra forma de hacer lo mismo pero un poco más elegante.

### **Crear prototipos con la nueva versión**

ExtJS mejoró la manera de crear prototipos en la versión 2.x, en esta versión ya no es necesario crear una función constructora y luego extenderla, simplemente utilizamos la propiedad “constructor” y extendemos de Object.

1. Ext.ns("com.quizzpot.tutorial");
- 2.

```

3. com.quizzpot.tutorial.Person = Ext.extend(Object,{ //Step 1
4. username : "none",
5. email : "none",
6. name : "none",
7. startTime : new Date(),
8. endTime : new Date(),
9.
10. constructor : function(options){ //Step 2
11. Ext.apply(this,options || {});
12.
13. console.debug("Person constructor!");
14. },
15.
16. checkIn : function(){
17. this.startTime = new Date();
18. },
19.
20. checkOut : function(){
21. this.endTime = new Date();
22. },
23.
24. workingHours: function(){
25. return (this.endTime - this.startTime)/1000;
26. }
27.
28. });

```

En el paso uno extendemos directamente de Object quien es el padre de todos los objetos, es importante mencionar que el método “Ext.extend” no modifica la superclase (superprototipo), sino que regresa un nuevo objeto tomando la super clase y añadiéndole las propiedades y métodos de loa subclase.

En el paso dos definimos el constructor de la clase (prototipo) y ahí alojamos el código que se ejecutará cuando se creen nuevas instancias.

Personalmente me gusta mucho esta nueva manera de hacer prototipos, es muy fácil y el código se lee muy bien, además de que nos evita la fatiga de asignar nosotros mismos cada propiedad y método al “prototype” de nuestro objeto.

### Sobre escritura de métodos

Hasta ahorita hemos creado un solo prototipo, lo que haremos ahora es extenderlo para crear uno nuevo y sobre escribir algunos métodos.

```

1. com.quizzpot.tutorial.Employee = Ext.extend(com.quizzpot.tutorial.Person,{ //Step 1
2. position : "Jr. Programmer",
3. technicalSkills : ["JavaScript","Ext JS","XHTML/CSS","Java"],
4. tasks : [],
5.
6. constructor : function(options){ //Step 2
7. console.debug("Employee constructor!");
8.
9. com.quizzpot.tutorial.Employee.superclass.constructor.call(this); //Step 3
10.
11. },
12.
13. checkOut : function(){
14. com.quizzpot.tutorial.Employee.superclass.checkOut.call(this); //Step 4
15. this.endTime.setHours(18);
16. return this.endTime;
17. },
18.

```

```

19. workingHours : function(){
20. var seconds = com.quizzpot.tutorial.Employee.superclass.workingHours.call(this);
21. return seconds/3600;
22. },
23.
24. assignTask : function(task){
25. this.tasks.push(task);
26. }
27. });

```

En el paso uno utilizamos el método Ext.extend para extender de la clase (prototipo) "Person" e iniciamos a escribir el código para la subclase "Employee".

En el paso dos creamos el constructor de "Employee" utilizando la propiedad "constructor" y automáticamente ExtJS creará la función constructora.

El paso tres es muy importante, aquí estamos ejecutando el constructor de la superclase, de esta manera es como reutilizaremos el código que ya hemos escrito en la superclase. Es importante mencionar que ExtJS crea automáticamente un acceso a la superclase mediante la propiedad "superclass" la cual es insertada a cada instancia, luego nos referimos al método que necesitamos ejecutar en la superclase y simplemente lo ejecutamos en el contexto actual usando el método "call".

En el paso cuatro estamos sobre escribiendo el método "checkout" pero también ejecutamos el "checkout" de la superclase que es el que define la hora de salida, luego en la subclase le asignamos una hora diferente, en este caso las seis de la tarde.

Podemos sobre escribir los métodos que necesitemos o bien crear nuevos métodos y propiedades para la subclase, ahora vamos a escribir la manera de crear instancias de la subclase:

```

1. var crysfel = new com.quizzpot.tutorial.Employee({
2. username : "crysfel",
3. email : "crysfel@dominio.com<script type="text/javascript">
4. /* <![CDATA[*/
5. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}catch(e){}})();
6. /*]]> */
7. </script>",
8. name : "Crysfel Villa"
9. });
10.
11. crysfel.checkIn();
12. setTimeout(function(){
13. crysfel.checkOut();
14. var time = crysfel.workingHours();
15. console.debug(time+" hours working!");
16. },3000);

```

Si ejecutas el ejemplo verás en la consola de tu navegador algunos mensajes, nota como se ejecuta primero el constructor de la subclase y luego éste manda ejecutar el constructor de la superclase.

## Conclusiones

ExtJS nos permite crear clases de una manera muy elegante, personalmente me agrada mucho esta manera ya que es muy entendible y fácil de realizar, en futuros tutoriales veremos cómo extender componentes de ExtJS para crear nuestras propias extensiones.

## Ejemplos variados

### Integración del FormPanel y el GridPanel

En esta ocasión vamos a Crear, Leer, Actualizar y Borrar (CRUD) un catalogo de contactos usando un formulario y mostrando los resultados en un “Grid” este es el que se encargara de hacer las operaciones antes mencionadas, con esto veremos otra aplicación básica de la integración de un FormPanel y el GridPanel, utilizando un Writer dentro del Store que maneja el Grid.

En este tutorial haremos un CRUD para poder editar la información contenida en el Grid, esto lo lograremos mediante el Writer de un store y usaremos un formulario para poder editar esa información. La siguiente imagen es un ejemplo de lo que tendremos al final de este tutorial.

Grid and Form Example

|           |                  |
|-----------|------------------|
| Name:     | Luiz             |
| Lastname: | Ortega           |
| E-mail:   | luiz@gmail.com   |
| Age:      | 19               |
| Phone:    | 123 - 456 - 7893 |
| Country:  | Mexico           |

| <input type="checkbox"/>            | Name      | Lastname | E-mail           | Phone            |
|-------------------------------------|-----------|----------|------------------|------------------|
| <input checked="" type="checkbox"/> | Luiz      | Ortega   | luiz@gmail.com   | 123 - 456 - 7893 |
| <input type="checkbox"/>            | Sasha     | Cohen    | sasha@gmail.com  | 123 - 456 - 7893 |
| <input type="checkbox"/>            | Cristiano | Ronaldo  | cr@gmail.com     | 123 - 456 - 7893 |
| <input type="checkbox"/>            | John      | Smith    | john@hotmail.com | 123 - 456 - 7893 |

Resultado Final

Durante el tutorial se irá explicando el código necesario para realizarlo, recuerda que puedes descargar el código fuente si te es necesario.

#### La Base de datos

La información para el Grid estará en una base de datos de MySQL que contendrá una tabla llamada “contacts”, en esta tabla solo tenemos la información básica de un contacto. El código para generar la tabla se muestra a continuación.

1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Servidor: localhost
6. -- Tiempo de generación: 28-10-2010 a las 16:30:53
7. -- Versión del servidor: 5.1.36
8. -- Versión de PHP: 5.3.0
- 9.

```

10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11.
12. --
13. -- Base de datos: `test`
14. --
15.
16. -----
17. --
18. --
19. -- Estructura de tabla para la tabla `contacts`
20. --
21.
22. CREATE TABLE IF NOT EXISTS `contacts` (
23. `id` int(11) NOT NULL AUTO_INCREMENT,
24. `email` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
25. `firstName` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
26. `lastName` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
27. `age` varchar(3) COLLATE utf8_unicode_ci NOT NULL,
28. `phone` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
29. `country` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
30. PRIMARY KEY (`id`)
31.) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT
 =5 ;
32.
33. --
34. -- Volcar la base de datos para la tabla `contacts`
35. --
36.
37. INSERT INTO `contacts` (`id`, `email`, `firstName`, `lastName`, `age`, `phone`, `country`) VALUE
 S
38. (1, 'luiz@gmail.com<script type="text/javascript">
39. /* <![CDATA[*/
40. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
 =parseInt(a.substr(0,2),16);for(j=2;a.length-
 j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
 ;l.parentNode.replaceChild(s,l);}catch(e){}})();
41. /*]]> */
42. </script>', 'Luiz', 'Ortega', '19', '123 - 456 - 7893', 'Mexico'),
43. (2, 'sasha@gmail.com<script type="text/javascript">
44. /* <![CDATA[*/
45. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
 =parseInt(a.substr(0,2),16);for(j=2;a.length-
 j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
 ;l.parentNode.replaceChild(s,l);}catch(e){}})();
46. /*]]> */
47. </script>', 'Sasha', 'Cohen', '25', '123 - 456 - 7892', 'USA'),
48. (3, 'cr@gmail.com<script type="text/javascript">
49. /* <![CDATA[*/
50. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
 =parseInt(a.substr(0,2),16);for(j=2;a.length-
 j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
 ;l.parentNode.replaceChild(s,l);}catch(e){}})();
51. /*]]> */
52. </script>', 'Cristiano', 'Ronaldo', '24', '123 - 456 - 7891', 'Portugal'),
53. (4, 'john@hotmail.com<script type="text/javascript">
54. /* <![CDATA[*/
55. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r
 =parseInt(a.substr(0,2),16);for(j=2;a.length-
 j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s)
 ;l.parentNode.replaceChild(s,l);}catch(e){}})();

```

```

56. /*]> */
57. </script>, 'John', 'Smith', '21', '123 - 456 - 7894', 'USA');

```

La base de datos se llama “test” pero puedes usar el nombre que te agrade, solo recuerda cambiar el nombre cuando se haga la conexión mediante PHP.

### Exponer la información

Como la información está almacenada en una base de datos, necesitamos realizar la conexión vía PHP o cualquier otro lenguaje de servidor y un simple “SELECT” de la siguiente manera. El siguiente código se debe escribir en un archivo llamado “serverside/getContacts.php”.

```

1. 1. <?php //step 1
2. $connection= mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
4.
5. $result= mysql_query("SELECT * FROM contacts"); //step 2
6.
7. $data= array();
8.
9. while($row= mysql_fetch_array($result)){ //step 3
10. array_push($data,array(
11. "id" => $row["id"],
12. "first" => $row["firstName"],
13. "last" => $row["lastName"],
14. "email" => $row["email"],
15. "age" => $row["age"],
16. "phone" => $row["phone"],
17. "country"=>$row["country"]
18.));
19. }
20.
21. echo json_encode(//step 4
22. array(
23. "success" => true,
24. "data" => $data
25.));

```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos se hace el query que regresa todo lo que contiene la tabla “contacts”, es un query muy sencillo.

En el paso tres se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo con la información de cada uno de los contactos.

En el paso cuatro se imprime la información en formato JSON, la respuesta será como el siguiente ejemplo:

```

1. {"success":true,"data":[{"id":"1","first":"Luiz","last":"Ortega","email":"luiz@gmail.com<script type="text/javascript">
2. /* <![CDATA[*/
3. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}catch(e){}})();
4. /*]> */
5. </script>","age":"19","phone":"123 - 456 - 7893","country":"Mexico"}, {"id":"2","first":"Sasha","last":"Cohen","email":"sasha@gmail.com<script type="text/javascript">
6. /* <![CDATA[*/

```

```

7. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}}catch(e){{}})();
8. /*]]> */
9. </script>,"age":"25","phone":"123 - 456 -
7892","country":"USA"},{"id":"3","first":"Cristiano","last":"Ronaldo","email":"cr@gmail.com<script type="text/javascript">
10. /* <![CDATA[*/
11. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}}catch(e){{}})();
12. /*]]> */
13. </script>,"age":"24","phone":"123 - 456 -
7891","country":"Portugal"}, {"id":"4","first":"John","last":"Smith","email":"john@hotmail.com<script type="text/javascript">
14. /* <![CDATA[*/
15. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}}catch(e){{}})();
16. /*]]> */
17. </script>,"age":"21","phone":"123 - 456 - 7894","country":"USA"}]

```

Esto es suficiente para poder generar la interface del Grid mediante JavaScript ya que tenemos toda la información necesaria.

### **Empaquetando el Tutorial**

Vamos a empaquetar el código para evitar conflictos con otras variables o librerías. Esto es algo muy importante y siempre lo debemos realizar.

```

1. Ext.ns("com.quizzpot.tutorials");
2.
3. com.quizzpot.tutorials.CrudForm = {
4. init : function() {
5. //code
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorials.CrudForm.init,com.quizzpot.tutorials.CrudForm);

```

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### **Creando el CRUD**

Lo que haremos a continuación es solicitar la información al servidor, esto será ya dentro de la función “init” de nuestro archivo “crudexample.js”. Como mencionamos anteriormente usaremos un “writer” que se encargue de hacer las operaciones básicas (leer, escribir, modificar y eliminar). Para lograr esto es necesario definir la propiedad “api” con las URLs donde el “proxy” realizará las peticiones Ajax de manera automática.

```

1. var proxy = new Ext.data.HttpProxy({ //step 1
2. api: {
3. read : "serverside/getContacts.php",
4. create : "serverside/createContact.php",
5. update : "serverside/updateContact.php",
6. destroy : "serverside/destroyContact.php"
7. }
8. });

```

En el paso uno se crea una instancia de “Ext.data.HttpProxy” esto es para que el “store” realice las peticiones Ajax de manera automática.

A continuación haremos el “Reader” y el “Writer”:

```
1. var reader = new Ext.data.JsonReader({ //step 1
2. totalProperty : 'total',
3. successProperty : 'success', // indica la propiedad que define si se ha insertado/actualizado o
 borrado con éxito
4. messageProperty : 'message',
5. idProperty : 'id',
6. root : 'data' //este es el nombre del parámetro que llega al servidor con el JSON modificado
 o
7. },
8. [
9. {name: 'first'},
10. {name: 'last'},
11. {name: 'email'}
12.]);
13. var writer = new Ext.data.JsonWriter({ //step 2
14. encode : true,
15. writeAllFields : true //decide si se manda al servidor solamente los campos modificados o todo
16. })
17.
18. this.store = new Ext.data.Store({
19. proxy : proxy,
20. reader : reader,
21. writer : writer,
22. autoSave : true
23. });
24. this.store.load();
```

En el paso uno lo que hacemos es crear una instancia del “JsonReader” este componente nos permitirá leer e interpretar la respuesta del servidor para cada petición Ajax realizada. En el paso tres se crea la instancia del “JsonWriter”.

Los tres fragmentos de código anteriores ya los hemos visto a detalle en tutoriales pasados, esa es la razón por la que no entramos a detalles de estructura solo hacemos mención algunos atributos. Si esto es nuevo para ti, te recomendamos revisar el tema sobre esto.[//link a lo onda del CRUD 😊](#)

### Creado el Grid

A continuación crearemos el Grid. Lo primero que haremos es desplegar la información que traemos desde la base de datos.

```
1. var sm = new Ext.grid.CheckboxSelectionModel(); //step 1
2.
3. this.grid = new Ext.grid.GridPanel({ //step 2
4. region : "center",
5. margins : "3 3 3 3",
6. width : 300,
7. store : this.store, // step 3
8. columns : [
9. sm,
10. {header:'Name', dataIndex:'first', sortable: true, width:80},
11. {header:'Lastname', dataIndex:'last', sortable: true, width:80},
12. {header:'E-mail', dataIndex:'email', sortable: true, width:100}
13.],
14. tbar : [//step 4
15. {text:"Delete", scope:this, handler:this.onDelete, iconCls:'delete-icon'}]
```

```

16.],
17. sm : sm,
18. border : false,
19. stripeRows : true
20. });

```

En el paso uno creamos una instancia del componente “CheckboxSelectionModel” para hacer que el selectionModel funcione mediante un checkbox.

En el paso dos se crea la instancia del GridPanel para formar la tabla que contendrá la información recibida de la base de datos.

Notemos que esta tabla cuenta con los dos atributos especiales “region” y “margins” el primero es para poder colocar nuestra tabla en una ventana que use un layout de tipo “border”. Por esta razón nuestro tabla tiene este atributo el cual nos dice que queremos que la tabla esté en el centro de la ventana. El segundo solo crea un margen alrededor del componente a la distancia en pixeles que le especificamos.

En el paso tres se le pasa a la tabla el “store” que contiene la información a desplegar en el grid.

En el paso cuatro se está agregando el botón “eliminar” en la barra de herramientas. Notemos que al botón se le está agregando una imagen para esto es necesario definir la clase CSS (delete-icon) en nuestra hoja de estilos o documento HTML de la siguiente manera:

```
1. .delete-icon{background:transparent url(Icons/delete.png) 0 0 no-repeat !important;}
```

Por el momento no trabajaremos con la función que está en el botón ya que lo haremos un poco más adelante así que puedes comentar el atributo “handler”. Por último colocaremos nuestra tabla en una ventana para poder mostrar los registros de esta.

```

1. var win = new Ext.Window({
2. title : "Grid and Form Example",
3. layout : "border",
4. width : 600,
5. height : 400,
6. items : [this.grid]
7. });
8. win.show();

```

Este es código ya muy común para nosotros. Solo mencionaremos algo relacionado con lo anterior, si notamos esta ventana tiene un layout de tipo “border” esto con el fin de acomodar los componentes en regiones. Por el momento solo contamos con nuestra tabla.

En estos momentos tendríamos algo semejante a esto:

The screenshot shows a window titled "Grid and Form Example". At the top left is a red "Delete" button with a trash icon. The main area is a grid table with the following data:

| <input type="checkbox"/> | Name      | Lastname | E-mail           | Phone Number     | Age | Country  |
|--------------------------|-----------|----------|------------------|------------------|-----|----------|
| <input type="checkbox"/> | Luiz      | Ortega   | luiz@gmail.com   | 123 - 456 - 7893 | 19  | Mexico   |
| <input type="checkbox"/> | Sasha     | Cohen    | sasha@gmail.com  | 123 - 456 - 7892 | 25  | USA      |
| <input type="checkbox"/> | Cristiano | Ronaldo  | cr@gmail.com     | 123 - 456 - 7891 | 24  | Portugal |
| <input type="checkbox"/> | John      | Smith    | john@hotmail.com | 123 - 456 - 7894 | 21  | USA      |

Colocando el Grid

### Creado el Formulario

En la sección pasada creamos la tabla que muestra la información de la base de datos, ahora haremos el formulario para introducir y editar la información.

Lo primero que haremos es crear el formulario.

```
1. this.form = new Ext.form.FormPanel({
2. region : "west",
3. width : 250,
4. bodyStyle : "padding: 10px;",
5. url : "serverside/updateContact.php", //step 1
6. margins : "3 3 3 3",
7. border : false,
8. defaults : {allowBlank: false},
9. items : [//step 2
10. {xtype : "textfield", name : "first", fieldLabel : "Name"},
11. {xtype : "textfield", name : "last", fieldLabel : "Lastname"},
12. {xtype : "textfield", name : "email", fieldLabel : "E-mail", vtype :"email"},
13. {xtype : "textfield", name : "age", fieldLabel : "Age"},
14. {xtype : "textfield", name : "phone", fieldLabel : "Phone"},
15. {xtype : "textfield", name : "country", fieldLabel : "Country"},
16. {xtype : "textfield", name : "id",hidden:true,allowBlank:true}
17.],],
18. fbar : [{text : "Update", scope : this, handler: this.onUpdate}, //step 3
19. {text : "Create", scope : this, handler: this.addContact}],
20.});
```

Lo primero que hacemos es crear la instancia del FormPanel.

Si notamos nuestro formulario cuenta con los atributos “region” y “margins” del mismo modo que en la sección anterior nuestra tabla. El tributo región coloca el formulario en la posición “oeste” y “margins” crea un margen alrededor del formulario.

En el paso uno específicas con el atributo “url” el archivo que manejara el evento “submit” del formulario, esto lo veremos un poco más adelante.

En el paso dos tenemos los campos que formarán el formulario, en general todos los “textfields” solo cuentan con un “nombre” y su “etiqueta”. Notemos el caso de el campo que recibe el correo electrónico que introduce el usuario, este cuenta con un “vtype” de tipo “email” esto es con el fin de validar que el usuario ingrese una dirección de correo electrónico válida al un cambio o crear un nuevo contacto.

El campo de nombre “id” también es un caso especial, si notamos este campo está oculto ya que lo definimos así por medio del atributo “hidden”, este campo se puede quedar en blanco. El por qué de esto lo explicaremos más adelante.

En el paso tres se crean los botones “actualizar” y “crear”. Por el momento no trabajaremos con las funciones que están en los botones ya que lo haremos un poco más adelante así que puedes comentar el atributo “handler”.

Ya que tenemos el formulario listo lo siguiente es agregarlo a la venta, para poder mostrarlo.

```
1. var win = new Ext.Window({
2. title : "Grid and Form Example",
3. layout : "border",
4. width : 600,
5. height : 400,
6. items : [this.grid,this.form] //step 1
7. });
8. win.show();
```

En el paso uno solo agregamos el formulario a la ventana por medio del atributo “items”, en este momento deberíamos tener algo como la siguiente imagen:

**Grid and Form Example**

|           |                      |                                       |  |  |
|-----------|----------------------|---------------------------------------|--|--|
| Name:     | <input type="text"/> | <input type="button" value="Delete"/> |  |  |
| Lastname: | <input type="text"/> |                                       |  |  |
| E-mail:   | <input type="text"/> |                                       |  |  |
| Age:      | <input type="text"/> |                                       |  |  |
| Phone:    | <input type="text"/> |                                       |  |  |
| Country:  | <input type="text"/> |                                       |  |  |

| <input type="checkbox"/> | Name      | Lastname | E-mail           | Phone        |
|--------------------------|-----------|----------|------------------|--------------|
| <input type="checkbox"/> | Luiz      | Ortega   | luiz@gmail.com   | 123 - 456789 |
| <input type="checkbox"/> | Sasha     | Cohen    | sasha@gmail.com  | 123 - 456789 |
| <input type="checkbox"/> | Cristiano | Ronaldo  | cr@gmail.com     | 123 - 456789 |
| <input type="checkbox"/> | John      | Smith    | john@hotmail.com | 123 - 456789 |

Colocando el Formulario

### Grabar los Registros Modificados

Hasta este punto tenemos la información en nuestro “Grid” lo siguiente que tenemos que hacer para poder cumplir una de las funciones del CRUD es editar los registros del “Grid” y guardarlos en la base de datos.

Para esto tenemos que sacar la información del Grid y desplegarlo en el formulario. Lo primero que haremos es desplegar la información del Grid en el formulario, con la información ya en el formulario la podremos editar y guardar en la base de datos.

Lo primero que haremos es obtener la información del contacto, esto lo haremos de la siguiente manera:

```
this.grid.on("rowClick",this.loadRecord,this);
```

Agregamos el evento “rowClick”, este evento se dispara cuando se da clic sobre un registro de la tabla, notemos que este evento es manejado por la función “loadRecord” es importante mencionar que también se le pasa el scope donde se ejecutará la función.

La función loadRecord lo que hace es hacer una petición al servidor con el “id” del contacto para traer la información de este y desplegarla en el formulario, esta función la veremos a continuación:

```

1. loadRecord : function(Grid,index,ev){ //step 1
2. var rec = this.grid.store.getAt(index); //step 2
3.
4. this.form.getForm().load({ //step 3
5. url : "serverside/fillForm.php", //archivo php que se encargará de hacer la petición al servidor
6.
7. params : {record:rec.id}, //step 4
8. failure : function(form, action){
9. Ext.Msg.alert("Error","Load failed");
10. }
11. },

```

En el paso uno se crea la función recibiendo los parámetros que nos manda el evento “rowClick”.

En el paso dos usamos el parámetro “index” el cual nos marca la fila en la que se encuentra el registro sobre el cual se dio clic. Esto lo hacemos con el fin de obtener la información de ese registro para poder cargarla en el formulario.

En el paso tres usamos el método “load” para cargar hacer la petición Ajax y desplegar el resultado en el formulario.

En el paso cuatro como parámetro de la petición mandamos el “id” del registro el cual obtuvimos en el paso uno.

Con esto es suficiente para hacer la petición al servidor, lo que sigue es crear el archivo PHP (serverside/fillForm.php) que es el que se comunicará con la base de datos.

```
1. <?php //step 1
2. $connection = mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection) or die("Error loading the DataBase".mysql_error());
4.
5. $id = $_POST["record"];
6. //step 2
7. $query = sprintf("SELECT * FROM contacts WHERE id =%d",
8. mysql_real_escape_string($id));
9.
10. $result = mysql_query($query);
11.
12. $info=array();
13. //step 3
14. if($row = mysql_fetch_array($result)){
15. $info = array(
16. "id" => $row["id"],
17. "first" => $row["firstName"],
18. "last" => $row["lastName"],
19. "email" => $row["email"],
20. "age" => $row["age"],
21. "phone" => $row["phone"],
22. "country"=> $row["country"]
23.);
24. }
25. //step 4
26. echo json_encode(array(
27. "success" => true,
28. "data" => $info
29.));
```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos, al recibir la información que se manda en el evento “load” del formulario, hacemos el query que regresa todo lo que contiene el registro con ese “id”, es un query muy sencillo.

En el paso tres se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo con la información del contacto.

En el paso cuatro se imprime la información en formato JSON, la respuesta será como el siguiente ejemplo:

```
1. {"success":true,"data":{"id":"2","first":"Sasha","last":"Cohen","email":"sasha@gmail.com<script typ
e="text/javascript">
2. /* <![CDATA[*/
```

```

3. (function(){try{var s,a,i,j,r,c,l=document.getElementById("__cf_email__");a=l.className;if(a){s="";r=parseInt(a.substr(0,2),16);for(j=2;a.length-j;j+=2){c=parseInt(a.substr(j,2),16)^r;s+=String.fromCharCode(c);}s=document.createTextNode(s);l.parentNode.replaceChild(s,l);}}catch(e){}})();
4. /*]]> */
5. </script>,"age":"25","phone":"123 - 456 - 7892","country":"USA"})

```

Con esto cuando demos clic sobre un registro de la tabla la información de ese contacto será cargada automáticamente en el formulario. Y deberíamos tener algo como la siguiente imagen:

|                                     | Name      | Lastname | E-mail           | Phone            |
|-------------------------------------|-----------|----------|------------------|------------------|
| <input type="checkbox"/>            | Luiz      | Ortega   | luiz@gmail.com   | 123 - 456 - 7892 |
| <input checked="" type="checkbox"/> | Sasha     | Cohen    | sasha@gmail.com  | 123 - 456 - 7892 |
| <input type="checkbox"/>            | Cristiano | Ronaldo  | cr@gmail.com     | 123 - 456 - 7892 |
| <input type="checkbox"/>            | John      | Smith    | john@hotmail.com | 123 - 456 - 7892 |

Llenando el Formulario

Ahora sí, la información está lista para poder ser editada, lo que debemos hacer ahora es guardar esa información. Para esto usaremos la función (onUpdate) que fue asignada al botón “actualizar”, esta función nos ayudará a realizar lo antes mencionado.

```

1. onUpdate : function(){
2. if (this.form.getForm().isValid()){ //step 1
3. this.form.getForm().submit({ //step 2
4. scope : this,
5. success : function(){
6. this.store.load(),
7. failure : function(response){
8. console.debug(response);
9. }
10. });
11. }else{
12. Ext.Msg.alert("Error","All fields are required");
13. }
14. },

```

En el paso uno verificamos que no se encuentre ningún campo vacío del formulario, de lo contrario se manda un mensaje de error.

En el paso dos se hace un “submit” al formulario, con el atributo “scope” indicamos el contexto donde se ejecutarán las funciones definidas en “success” y “failure”.

En la función definida para “success” lo único que hacemos es cargar el store con el propósito de actualizar la tabla con los cambios realizados a la información del contacto.

En la función “failure” solo se imprime en consola la información para ver donde se originó el problema.

Lo que debemos hacer ahora es crear el script PHP que va a actualizar la información en la base de datos, si recordamos en el formulario en el atributo “url” tenemos el archivo “serverside/updateContact.php”, este archivo se encarga de actualizar la base de datos.

```
1. <?php //step 1
2. $connection=mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection) or die("Error loading the DataBase".mysql_error());
4.
5. //step 2
6. $name = htmlentities($_POST["first"], ENT_QUOTES);
7. $last = htmlentities($_POST["last"], ENT_QUOTES);
8. $mail = htmlentities($_POST["email"], ENT_QUOTES);
9. $age = htmlentities($_POST["age"], ENT_QUOTES);
10. $phone = htmlentities($_POST["phone"], ENT_QUOTES);
11. $country = htmlentities($_POST["country"], ENT_QUOTES);
12. $id = $_POST["id"];
13.
14. //step 3
15. $query = sprintf("UPDATE contacts SET email = '%s', firstName = '%s', lastName = '%s', age =
16. '%s', phone = '%s', country = '%s' WHERE id=%d",
17. mysql_real_escape_string($mail),
18. mysql_real_escape_string($name),
19. mysql_real_escape_string($last),
20. mysql_real_escape_string($age),
21. mysql_real_escape_string($phone),
22. mysql_real_escape_string($country),
23. mysql_real_escape_string($id));
24.
25. $rs = mysql_query($query);
26.
27. //step 4
28. echo json_encode(array(
29. "success" => mysql_errno() == 0,
30. "msg" => mysql_errno() == 0?"Contact inserted successfully":mysql_error()
30.));
```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos lo que hacemos es obtener la información mandada por el submit, notemos que la información está siendo pasada por la función PHP “htmlentities” que elimina las etiquetas html en la información que fue introducida. Lo que esta función hacer es convertir la siguiente etiqueta html **<b>Luiz</b>** a algo como lo siguiente **<b>Luiz</b>**

En el paso tres hacemos un “update” a la tabla en la que están guardados nuestros contactos y le damos como valores la información que fue mandada por el “Store”.

En el paso cuatro lo que hacemos es mandar un mensaje de éxito si es que todo sale bien, de lo contrario se manda el mensaje de error de MySQL.

Hasta este momento ya cumplimos con dos de las cuatro funciones de un CRUD leer y actualizar.

## Crear un Registro Nuevo

En la sección anterior logramos guardar los registros que fueron modificados por el usuario, ahora vamos a insertar nuevos registros en el Grid, para esto usaremos la función (addContact) que fue asignada al botón “crear” esta función nos ayudara a realizar lo antes mencionado.

```
1. addContact : function(){
2. if (this.form.getForm().isValid()){ //step 1
3. var contact = new this.store.recordType({ //step 2
4. first : this.form.getForm().getValues().first,
5. last : this.form.getForm().getValues().last,
6. email : this.form.getForm().getValues().email,
7. country : this.form.getForm().getValues().country,
8. phone : this.form.getForm().getValues().phone,
9. age : this.form.getForm().getValues().age
10. });
11. this.store.insert(0,contact); //step 3
12. }else{
13. Ext.Msg.alert("Error","All fields are requiered");
14. }
15. }
```

Creamos la función dentro del objeto principal y en el paso uno verificamos que no se encuentren campos vacíos en el formulario, de lo contrario se manda un mensaje de error.

En el paso dos hacemos un registro nuevo asignándole en los campos del Grid la información que el usuario introduce en el formulario para este nuevo registro.

En el paso tres insertamos el contacto nuevo para amostrarlo en el Grid, al tener un “Writer” configurado automáticamente se realizará una petición al servidor con la información a grabar.

Lo siguiente que debemos hacer es guardar ese contacto nuevo en la base de datos. Para esto necesitamos crear el archivo “serverside/createContact.php”, el cual se encargará de introducir el nuevo registro a la base de datos.

```
1. <?php //step 1
2. $connection = mysql_connect("localhost", "root", "") or die("Connection Failed".mysql_error());
3. mysql_select_db("test", $connection) or die("Error loading the DataBase".mysql_error());
4.
5. $info = $_POST["data"];
6.
7. $data = json_decode(stripslashes($info)); //step 2
8.
9. //step 3
10. $name = htmlentities($data->first, ENT_QUOTES);
11. $last = htmlentities($data->last, ENT_QUOTES);
12. $email = htmlentities($data->email, ENT_QUOTES);
13. $age = htmlentities($data->age, ENT_QUOTES);
14. $phone = htmlentities($data->phone, ENT_QUOTES);
15. $country= htmlentities($data->country, ENT_QUOTES);
16.
17. //step 4
18. $query = sprintf("INSERT INTO contacts (email,firstName,lastName,age,phone,country) values ('%s','%s','%s','%s','%s','%s')",
19. mysql_real_escape_string($email),
20. mysql_real_escape_string($name),
21. mysql_real_escape_string($last),
22. mysql_real_escape_string($age),
23. mysql_real_escape_string($phone),
24. mysql_real_escape_string($country));
```

```

25.
26. $rs = mysql_query($query);
27.
28. //step 5
29. echo json_encode(array(
30. "success" => mysql_errno() == 0,
31. "msg" => mysql_errno() == 0?"Contact inserted successfully":mysql_error(),
32. "data" => array(
33. array(
34. "id" => mysql_insert_id(),
35. "first" => $name,
36. "last" => $last,
37. "email" => $email,
38. "age" => $age,
39. "phone" => $phone,
40. "country"=>$country
41.)
42.)
43.));

```

En el paso uno se realiza la conexión a la base de datos.

En el paso dos, una vez que recibimos la información que mando el “Store” lo decodificamos con la función json\_decode() de PHP con esto ya podemos acceder a la información que fue editada en el formulario, la función “stripslashes” elimina el carácter “\” que Apache le agrega, es posible que la configuración de tu servidor no lo haga, pero es bueno prevenir esto para evitarnos errores en el futuro.

En el paso tres revisamos que la información que recibimos del formulario no tenga etiquetas HTML con la función php htmlentities().

En el paso cuatro hacemos un “insert” y damos como valores la información que fue mandada por el formulario.

En el paso cinco lo que hacemos es mandar un mensaje de éxito si es que todo sale bien, de lo contrario se manda el mensaje de error de MySQL.

Es muy importante mencionar que el Json\_encode se está haciendo después del “success” y el “msg” mandamos la información que fue insertada a la base de datos, es muy importante mandar el “id” del nuevo registro, esto nos permitirá que los métodos update y delete funcionen correctamente, si no lo hacemos no funcionarán.

Con esto ya podremos insertar un nuevo registro en nuestro “Grid” y en nuestra base de datos.

### **Eliminando un Registro**

En estos momentos solo hemos hecho lo necesario para poder leer los registros y desplegarlos en nuestro “Grid”, podemos añadir nuevo registros y editar los registros existentes por medio de nuestro formulario. Lo único que nos hace falta es eliminar registros y con esto tendríamos las cuatro funciones del CRUD.

Para esto crearemos la función a la que hace referencia el botón “eliminar”, lo que haremos es lo siguiente:

1. onDelete : function(){ //step 1
2. var rows = this.grid.getSelectionModel().getSelections(); //step 2
3. if(rows.length === 0){ //step 3
4. return false;
5. }
6. this.store.remove(rows); //step 4
7. },

En el paso uno creamos la función “onDelete” dentro del objeto principal.

En el paso dos tomamos la instancia del “SelectionModel()” de nuestro “Grid”, con esto podemos obtener la información de las filas seleccionadas por el usuario mediante el método “getSelections()”.

En el paso tres lo que hacemos es verificar que el usuario haya seleccionado algo, si no hay nada seleccionado o la selección del “Grid” está vacía no se eliminará ningún registro ya que hacemos un return.

En el paso cuatro ya que se verificó que es un registro que puede ser borrado y lo eliminamos del “Grid” usando el método “remove” del store que contiene los records.

Lo siguiente que debemos hacer es crear el archivo(serverside/destroyContact.php”) que se encargará de borrar el registro de la base de datos.

```
1. <?php //step 1
2. $connection=mysql_connect("localhost","root","");
3. mysql_select_db("test",$connection) or die("Error loading the DataBase".mysql_error());
4.
5. $id = json_decode(stripslashes($_POST["data"])); //step 2
6.
7. //step 3
8. $query = sprintf("DELETE FROM contacts WHERE id = %d",
9. mysql_real_escape_string($id));
10.
11. $rs = mysql_query($query);
12.
13. //step 4
14. echo json_encode(array(
15. "success" => mysql_errno() == 0,
16. "msg" => mysql_errno() == 0?"Contact deleted successfully":mysql_error()
17.));
```

En el paso uno lo que hacemos es crear la conexión a la base de datos.

En el paso dos obtenemos la información que nos fue mandada, la decodificamos con Json\_decode() y la guardamos en la variable “id”. En este caso solo se nos envió el “id” del registro que se quiere eliminar.

En el paso tres solo hacemos el query para eliminar el registro.

En el paso cuatro mandamos como respuesta un mensaje de éxito y si ocurrió un error se manda el error como respuesta.

Si intentamos eliminar un registro notaremos que esta vez también fue borrado de nuestra base de datos. Esta sería la respuesta del servidor:

```
1. {"success":true,"msg":"Contact deleted successfully"}
```

Con esto terminamos la última parte de nuestro CRUD que es eliminar un registro.

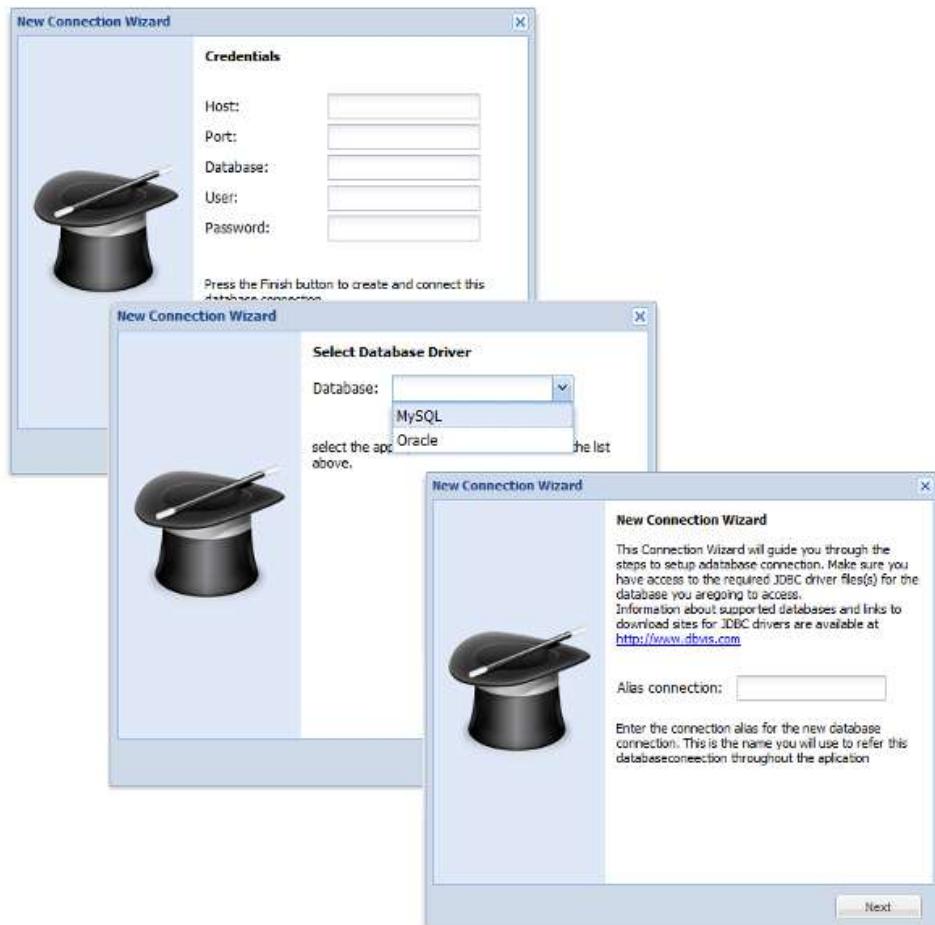
## Conclusión

En esta ocasión vimos como podemos integrar un formulario y una tabla para poder realizar algunos cambios a la información que esta contiene, con esto podemos crear una nueva aplicación de una manera más dinámica.

## Creando un Wizard

En esta ocasión veremos cómo podemos crear un Wizard, esto lo haremos utilizando un layout de tipo "Card" y un solo formulario, el wizard que simularemos recojerá la información necesaria para crear una conexión a una base de datos.

En este tutorial veremos la manera de cambiar la información que queremos desplegar en una ventana, esto lo haremos usando un "CardLayout" el cual nos permite cambiar el contenido de una ventana sin tener que actualizar en nuestro explorador, la siguiente imagen el resultado que se obtendrá al final del tutorial.



Resultado Final

### Empaquetando el Tutorial

Vamos a empaquetar el código para evitar conflictos con otras variables o librerías que usemos en nuestro proyecto.

```
1. Ext.ns("com.quizzpot.tutorial");
2.
3. com.quizzpot.tutorial.Wizard = {
4. index : 0,
5.
6. init : function(){
7. //code here
8. }
9. }
10. Ext.onReady(com.quizzpot.tutorial.Wizard.init,com.quizzpot.tutorial.Wizard);
```

Más adelante se explica la propiedad “index” que hemos definido dentro del objeto principal.

### Creando el Cardlayout

Lo primero que haremos es crear un formulario el cual contendrá la información que deseamos desplegar a lo largo de nuestro wizard. Dentro de la función “init” colocaremos el siguiente código:

```
1. this.form = new Ext.FormPanel({ // step 1
2. layout : "card", //step2
3. border : false,
4. activeItem : this.index, //step3
5. items : [this.createAllias(), this.createDriver(), this.createCredentials()] //step 4
6. });

});
```

En el paso uno solo hicimos una instancia del “FormPanel”.

Lo interesante viene en el paso dos, a estas alturas del curso los layouts que más hemos usado son de tipo “fit” y “border”, en esta ocasión usaremos uno de tipo “card”, este layout nos permite desplegar todos los “items” de esta instancia uno a la vez, además de proporcionarnos métodos para cambiar entre una “carta” y otra.

En el paso tres lo que hacemos con “activeItem” es indicar cual objeto del arreglo de “items” será el que se usará cuando se inicie la aplicación, y si recordamos “index” es igual a cero con esto indicamos que queremos el objeto de la posición cero del arreglo “items”.

En el paso cuatro al arreglo “items” le asignamos tres funciones “createAllias()”, “createDriver()” y por último “createCredentials()”, vamos a ir viendo en detalle cada una de estas funciones cuyo objetivo es crear la interface para cada paso del wizard.

### Definición de los pasos en el Wizard

La primera función será nuestra primera pantalla en la cual debe aparecer un campo donde el usuario asignará un nombre a la conexión, también tendremos una pequeña descripción para asistir al usuario.

La segunda función será la pantalla siguiente esta mostrará un combo en el cual el usuario seleccionará el servidor de base de datos, por ejemplo nuestro combo puede tener valores como: MySQL, Oracle Thin, SQL Server, Informix, etc. Esta información la obtendremos desde una tabla de una base de datos.

La última función creará la última pantalla en la que se capturan las credenciales del servidor, vamos a desplegar algunos campos de texto y uno de tipo password.

Por el momento podemos eliminar o comentar las últimas dos funciones, trabajaremos con ellas un poco más adelante.

### Función “createAllias”

Como mencionamos antes, esta función lo único que contiene es una pequeña descripción y un formulario para obtener el nombre de la conexión, esto lo haremos de la siguiente manera:

```
1. createAllias : function(){
2. return {
3. xtype : "panel",
4. layout : "form",
5. border : false,
6. padding : 10,
7. items : [{html : "<h1>New Connection Wizard</h1>
 "+
8. "<p>This Connection Wizard will guide you through the steps to setup a"+
9. "database connection. Make sure you have access to the required JDBC driver files(s) for the dat"+
10. "abase you are">"+
10. "going to access.
The information about supported databases and links to download sites for"+
10. "JDBC drivers are available"+
```

```

11. " at <a href)+"http://www.dbvis.com"+>http://www.dbvis.com</p>

", border : false},
12.
13. {xtype : "textfield", name : "alias"},
14. {html : "<p>
Enter the connection alias for the new data base connection. This is the
 name you will use to refer this database"+
15. "connection throughout the application</p>", border : false}]
16.
17. };
18. }

```

Lo que hicimos es crear la función donde en el “return” creamos un panel (utilizando su “xtype”) el cual tiene un layout de tipo “form” esto es para que al definir el textfield pueda aparecer correctamente su “Label”, de lo contrario no aparecerá.

Al panel que regresamos le podemos colocar todo lo que deseamos que aparezca en pantalla en el atributo “items”. Con esto tendríamos la primera pantalla lista pero no la podemos ver porque no se ha renderizado el formulario que la contiene.

Para poder ver nuestra pantalla lo que haremos es crear una ventana que tendrá al “form” principal, también colocaremos los botones que usaremos para el cambio entre pantallas.

```

1. //step 1
2. this.backBtn = new Ext.Button({text : "Back",handler : this.back,scope : this,hidden:true});
3. this.nextBtn = new Ext.Button({text : "Next",handler : this.next,scope : this});
4. this.finishBtn = new Ext.Button({text : "Finish", hidden : true, handler:this.finish,scope:this});
5.
6. this.win = new Ext.Window({
7. title : "New Connection Wizard",
8. layout : "fit",
9. width : 450,
10. height : 400,
11. resizable : false, //step 2
12. bodyCssClass : "wizard-image", //step 3
13. fbar : [this.backBtn,this.nextBtn,this.finishBtn], //step 4
14. items : [this.form] //step 5
15. });

```

En el paso uno creamos los tres botones que usaremos para el cambio entre pantallas, si notamos los botones “backBtn” y “finishBtn” están ocultos (“hidden”) ya que a estos botones les daremos un uso especial más adelante, si lo deseas puedes comentar la propiedad “handler” ya que no hemos definido las funciones todavía, más adelante lo haremos.

En el paso dos hacemos que no se puedan cambiar las dimensiones de la ventana.

En el paso tres se asigna la clase CSS “wizard-image” al cuerpo de la ventana, esta clase coloca una imagen a la ventana, para esto es necesario definir la clase “wizard-image” en una hoja de estilos o en el documento HTML de la siguiente manera:

```
1. .wizard-image{background:transparent url(wizard.png) 10px center no-repeat;padding-left:150px;}
```

Nótese que se ha asignado un padding de 150px a la imagen, esto es para que se despliegue correctamente.

En el paso cuatro se han asignado los botones al “footer bar” (fbar), esto permite posicionarlos en la parte inferior de la ventana.

El paso cinco solamente le asignamos el formulario a la ventana como contenido principal.

Por último es necesario renderizar los componentes que hemos creado, al usar una ventana solamente es necesario ejecutar el método “show” de la instancia “win”.

1. this.win.show();



Primera Pantalla

#### Función “createDriver”

Lo siguiente es crear la función “createDriver” que usaremos para la siguiente pantalla como mencionamos antes en esta pantalla el usuario escoge los drivers que usará, esta información será obtenida desde nuestra base de datos y desplegada en un combo. Primero veremos la estructura de la base de datos.

```
1. -- phpMyAdmin SQL Dump
2. -- version 3.2.0.1
3. -- http://www.phpmyadmin.net
4. --
5. -- Servidor: localhost
6. -- Tiempo de generación: 19-10-2010 a las 03:58:23
7. -- Versión del servidor: 5.1.36
8. -- Versión de PHP: 5.3.0
9.
10. SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
11.
12. --
13. -- Base de datos: `test`
14. --
15.
16. -----
17. --
18. --
19. -- Estructura de tabla para la tabla `drivers`
20. --
21.
```

```

22. CREATE TABLE IF NOT EXISTS `drivers` (
23. `id` int(11) NOT NULL AUTO_INCREMENT,
24. `database` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
25. `driver` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
26. PRIMARY KEY (`id`)
27.) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=
28. 3;
29.
30. -- Volcar la base de datos para la tabla `drivers`
31. --

```

Tomando en cuenta la definición de la tabla anterior necesitamos exponer la información usando nuestro lenguaje favorito, para este tutorial usaremos PHP, pero bien pueden usar Java, Ruby, Grails o cualquier otro lenguaje.

```

1. <?php
2. $link=mysql_connect("localhost", "root", "") or die ("Error conectando a la base de datos.");
3. mysql_select_db("test",$link) or die("Error seleccionando la base de datos.");
4.
5. $result=mysql_query("SELECT * FROM drivers");
6.
7. $drivers=array();
8. while($row=mysql_fetch_array($result)){
9. array_push($drivers,array(
10. "id" =>$row["id"],
11. "database" =>$row["database"],
12. "driver" =>$row["driver"]
13.));
14. }
15.
16. echo json_encode(
17. array(
18. "success" => true,
19. "info" => $drivers
20.)
21.);

```

El resultado al ejecutar ese código será el siguiente:

```

1. {"success":true,"info":[{"id":"1","database":"MySQL","driver":"com.mysql.jdbc.Driver"},{"id":"2","dat
abase":"Oracle","driver":"oracle.jdbc.driver.OracleDriver"}]}

```

Ya que tenemos la información necesaria lo siguiente es crear el combo y el panel que tendrá la información para la desplegar en pantalla. Lo haremos de la siguiente manera:

```

1. createDriver : function(){
2. //se crea el store
3. var store = new Ext.data.JsonStore({ //step 1
4. url :"drivers.php", //archivo donde sacará la información
5. root : "info",
6. fields : [//campos del store que usará
7. {name :"id"},
8. {name :"database", type : "string"},
9. {name :"driver", type : "string"},
10.]
11. });
12.
13. //se crea el combo asignándole el store
14. var combo = new Ext.form.ComboBox({ //step 2

```

```

15. fieldLabel :"Database",
16. name :"database",
17. forceSelection : true, // esta opción obliga al usuario a seleccionar un valor del combo
18. store : store, //asignándole el store
19. emptyText :"pick one DB...", // texto mostrado antes de que se seleccione algo
20. triggerAction :"all", // indica que siempre muestre todos los datos de su store
21. editable : false, // no se puede editar el contenido
22. border : false,
23. width : 150,
24. displayField : "database", //la información que mostrara dentro del combo
25. valueField : "id" // lo que enviará al servidor
26.);
27.
28. return { //step 3
29. xtype : "panel",
30. layout : "form",
31. padding : 10,
32. labelWidth : 60,
33. border : false,
34. items : [{html : "<h1>Select Database Driver</h1>
",border : false},
35. combo,
36. {html : "

<p>Select the appropriate database driver from the list above.</p>",
37. border : false}]
37. }
38. },

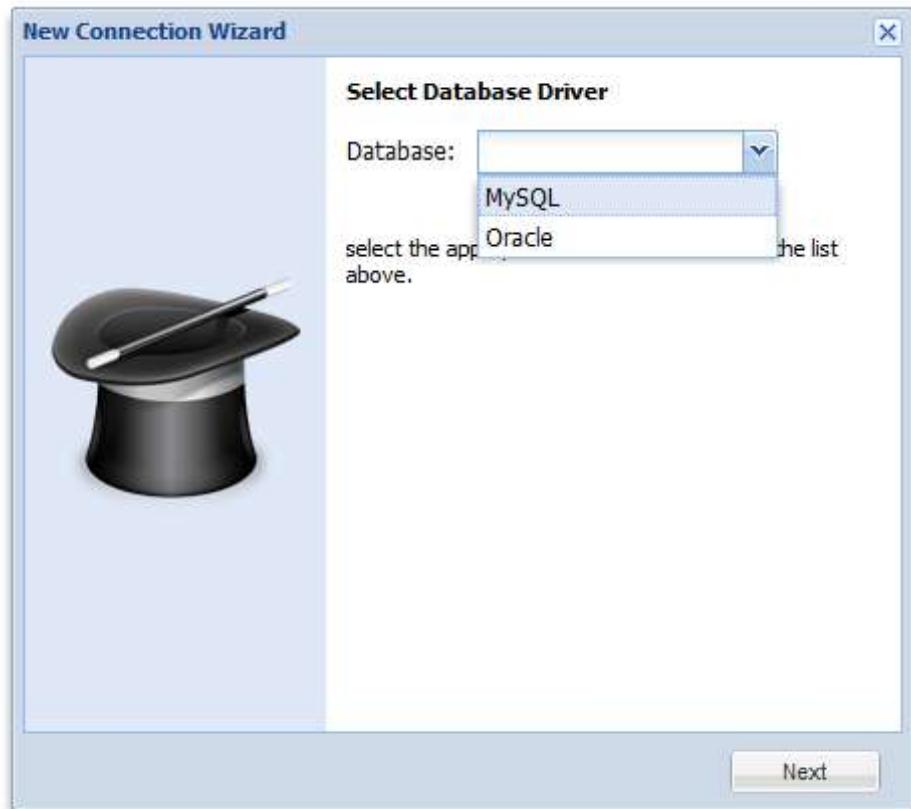
```

En el paso uno lo que hicimos fue crear el “store” que nos trae la información que obtenemos en el archivo driver.php, además de que lo configuramos para que pueda interpretar correctamente esa información.

En el paso dos creamos el combo con la información que contiene el store, en el código he comentado algunas configuraciones usadas.

En el paso tres creamos el panel con un layout de tipo “form” y en “items” le pasamos lo que queremos que contenga este panel, en primer lugar ponemos una descripción usando el atributo “html” luego pasamos el combo y por último se ponen las indicaciones también con un atributo “html”.

Con esto tendríamos la segunda pantalla que es parecida a esta.



Segunda pantalla

### Función “createCredentials”

Por ultimo crearemos el formulario donde el usuario introduce el nombre del servidor, el puerto, la base de datos, el usuario y la contraseña, esto será muy similar a las funciones anteriores:

```
1. createCredentials : function(){
2. return{
3. xtype : "panel",
4. layout : "form",
5. labelWidth : 100,
6. padding : 10,
7. border : false,
8. items : [{html : "<h1>Test</h1>

", border : false},
9. {fieldLabel : "Server",xtype : "textfield", name : "server"},
10. {fieldLabel : "Port",xtype : "textfield", name : "port"},
11. {fieldLabel : "Database",xtype : "textfield", name : "database"},
12. {fieldLabel : "UserID",xtype : "textfield", name : "user"},
13. {fieldLabel : "Password",xtype : "textfield", inputType : "password", name : "passwd"},
14. {html : "

<p>Press the Finish button to create and connect this database connection.</p>", border : false}]
15. }
16. },
```

Esta función básicamente funciona como las últimas dos. Lo que hacemos es crear el panel con un layout de tipo “form” y dentro de “items” colocamos lo que contendrá el panel, como lo que deseamos obtener es información que proporciona el usuario usamos “textfield” para que nos crea el campo donde el usuario podrá introducir la información, notemos que en el último “textfield” usamos “inputType” de tipo “password” con la finalidad de que este campo no muestre los caracteres alpha numericos ya que está destinado a para recibir una contraseña.

Con esto tenemos la tercera pantalla:



Tercera pantalla

#### Avanzar en los pasos del wizard

Ya que tenemos nuestras pantallas listas lo siguiente es asignarle acciones a los botones que definimos al inicio del tutorial, descomenta la propiedad "handler" de cada botón.

Para avanzar entre las pantallas crearemos la función "next", de la siguiente manera:

```
1. next: function(){
2. this.backBtn.show(); //step 1
3. if(this.index < this.form.items.length-1){ //step 2
4. this.index++;
5. var cardlayout = this.form.getLayout(); //step 3
6. cardlayout.setActiveItem(this.index);
7.
8. //step 4
9. if(this.index == this.form.items.length-1){ //si esta en el ultima carta
10. this.nextBtn.hide();
11. this.finishBtn.show();
12. }
13. }
14. },
```

En el paso uno simplemente mostramos el botón "back", esto es porque de inicio está oculto y lo que necesitamos es que se visualice cuando el usuario avanza a la segunda pantalla.

En el paso dos solamente se verifica que no sea el último paso del wizard, la propiedad "this.index" es un contador que nos servirá para saber en cual paso esta el usuario, si no es el ultimo paso entonces se incrementa la propiedad "index" en uno.

El paso dos es importantísimo, aquí tomamos el “CardLayout” que estamos usando para el wizard, el método “getLayout” regresa la instancia del layout que estamos usando. Una vez que tenemos el layout podemos asignarle una nueva carta usando el método “setActiveItem”.

En el paso cuatro revisamos si al cambiar de carta se está desplegando la última, de ser así se ocultará el botón “next” y se mostrará el botón “finish”.

### Retroceder en los pasos del wizard

Hasta este punto podemos avanzar en el wizard y llegar al final, ahora vamos a darle la opción al usuario de poder retroceder en los pasos del wizard, para eso creamos la función “back” de la siguiente manera.

```
1. back : function(){
2. if(this.index>0){ //step 1
3. this.index--;
4. var cardlayout = this.form.getLayout();
5. cardlayout.setActiveItem(this.index);
6. }
7.
8. //step 2
9. if(this.index == 0){ //si está en la primera carta
10. this.backBtn.hide();
11. }else{ //step 3
12. this.finishBtn.hide();
13. this.nextBtn.show();
14. }
15. },
```

En el paso uno verificamos que el usuario no esté visualizando el paso número uno del wizard ya que si estamos al inicio no es posible seguir retrocediendo, si “index” es mayor a cero entonces si es posible retroceder y cambiamos la pantalla.

En el paso dos revisamos que si al retroceder el usuario esta viendo el primer paso entonces ocultamos el botón “back”, ya que no es necesario seguir mostrándolo.

El paso tres se ejecuta siempre y cuando no estemos en el primer paso, aquí ocultaremos el botón “finish” y mostraremos el botón “next” ya que si se ha presionado el botón “back” es seguro que ya no estamos en el ultimo paso.

### Finalizando el wizard

Ya tenemos la funcionalidad básica, el usuario puede avanzar y retroceder entre los pasos que tiene el wizard, ahora tenemos que programar lo que necesitamos realizar cuando el usuario ha concluido con el wizard, esto es cuando da click sobre el botón “Finish”, para esto creamos el handler del botón de la siguiente manera:

```
1. finish : function(){
2. this.form.getForm().submit({
3. url : "createConnection.php",
4. scope : this,
5. success : this.msg,
6. failure : this.msg
7. });
8. },
```

Aquí puede ir el código que deseas, para este ejemplo solamente se le ha hecho un “submit” al formulario principal enviando la información al archivo “createConnection.php” el cual debería poder gestionar la información recibida y procesarla correctamente, una vez que responde el servidor se ejecutará la función “msg” la cual será algo como lo siguiente:

```
1. msg : function(){
2. Ext.Msg.alert("Alert","Your connection has been created succesfully.");
```

```
3. this.win.close();
4. }
```

Aquí también puedes poner lo que deseas, en este caso solamente se despliega un mensaje y además se cierra la ventana del wizard automáticamente.

### Conclusion

En este tutorial vimos algunos puntos importantes como la creación de un “CardLayout”, también se discutió sobre la manera de moverse sobre las cartas creadas y algo que quiero resaltar es la manera en como usamos un solo formulario para todos los campos que había en las cartas, esto nos permite recolectar la información de una manera extremadamente sencilla.

## Editando una Imagen

En esta ocasión veremos como interactuar con una imagen por medio de sliders, los cuales nos ayudaran a que este elemento pueda ser editado. Veremos cómo cambiar propiedades básicas de nuestro elemento.

En este tutorial vamos a modificar mediante los sliders la opacidad y dimensiones de una imagen, esto nos permitirá comprender el uso adecuado de componente “Ext.slider.SingleSlider”, la siguiente imagen es un ejemplo de lo que obtendremos al final del tutorial:



Resultado Final

### Empaquetando el Tutorial

Vamos a empaquetar el código para evitar conflictos con otras variables.

```
1. Ext.ns("com.quizzpot.tutorial");
2. com.quizzpot.tutorial.SliderTutorial= {
3. init : function(){
4. //code
5. }
6. }
7. Ext.onReady(com.quizzpot.tutorial.SliderTutorial.init, com.quizzpot.tutorial.SliderTutorial);
```

### Colocando la Imagen

Lo primero que haremos es crear la estructura html sobre la cual trabajaremos a lo largo del tutorial.

```
1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3.
4. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```

7. <title>Demo: Slider | Quizzpot</title>
8.
9. <link rel="stylesheet" type="text/css" href="../ext-3.2.1/resources/css/ext-all.css" />
10. <link rel="stylesheet" type="text/css" href="../resources/style.css" />
11.
12. <script type="text/javascript" src="../ext-3.2.1/adapter/ext/ext-base.js"></script>
13. <script type="text/javascript" src="../ext-3.2.1/ext-all.js"> </script>
14. <script type="text/javascript" src="slider.js"></script>
15.
16. <style type="text/css">
17. #frame{width:800px;height:400px;margin:50px;}
18. #canvas{width:595px;height:100%;overflow:hidden;border:1px solid #99BBE8;float:left;}
19. #tools{width:200px;height:100%;float:right;}
20. </style>
21.
22. </head>
23. <body>
24.
25. <div id="frame">
26. <div id="canvas">
27.
28. </div>
29. <div id="tools">
30. </div>
31. </div>
32.
33. </body>
34. </html>

```

Lo que hicimos fue crear tres divs, el primero es “frame” este es el div en el cual se encontraran alojados los otros divs que es “canvas” en el cual colocaremos la imagen que editaremos en el tutorial, y por ultimo está el div “tools” este lo usaremos para renderizar el panel con los slider.

También creamos estilos para nuestros divs con estos estilos posicionamos y les asignamos un tamaño. Si ejecutamos el html podemos ver que solo tenemos los divs y la imagen, lo primero que haremos es tomar la imagen por su id, el código que aparece a continuación debe ir en el archivo slider.js dentro de la función init.

```

1. var el = Ext.get('img');//step 1
2. frame = Ext.get("canvas"); //step 2
3.
4. el.setWidth("50%"); //step 3
5. setTimeout(function(){
6. el.center(frame);
7. },500);

```

En el paso uno lo que hacemos es obtener la imagen por medio de su id el cual está definido dentro de nuestro documento html.

En el paso dos obtenemos el div donde está colocada la imagen.

En el paso tres cambiamos el tamaño de la imagen usando “setWidth()”.

Usamos “setTimeout()” para ejecutar la función después del tiempo marcado en milisegundos, esto da tiempo para que a se pueda calcular el nuevo tamaño de la imagen.

Y por último usamos “center()” para colocar la imagen en el centro, si notamos le pasamos como parámetro “frame” esta causara que se centre dentro del div “canvas” que es el espacio que dedicamos para colocar nuestra imagen.



Colocando la Imagen

### Creando los Sliders

Ya que tenemos nuestra imagen lista, lo siguiente es colocar los sliders “size” y “opacity” con los cuales podremos manipular el tamaño y la trasparencia de nuestra imagen, lo haremos de la siguiente manera:

```
1. var size = new Ext.slider.SingleSlider({ //step 1
2. fieldLabel : "Size",
3. width : 100, // el tamaño que tendrá el slider
4. minValue : 20, // el valor mínimo que se obtendrá del slider
5. maxValue : 120, //el valor máximo que obtendrá el slider
6. value : 50, //valor inicial que tendrá el slider
7. plugins : tip
8. });
9.
10. var opacity = new Ext.slider.SingleSlider({ //step 2
11. fieldLabel : "Opacity",
12. width : 100,
13. increment : 10, // el slider tendrá un aumento de 10 en 10
14. minValue : 0,
15. maxValue : 100,
16. value : 100,
17. plugins : tip //step 3
18. });
```

En el paso uno creamos el slider “size”, y si vemos en el paso dos se crea el slider “opacity” ambos tienen los mismos atributos solo cambia el atributo “imcremenent” que se encuentra en “opacity”.

Notemos que estamos usando un “SingleSlider” ya que este componente nos permite renderizarlo verticalmente si es que lo deseamos.

En el paso tres lo que hacemos es pasarle los “tips” a los sliders vía “plugins” pero para esto tenemos que crearlos antes de crear los sliders, lo haremos dela siguiente manera:

```
1. Ext.QuickTips.init(); //step 1
2.
3. var tip = new Ext.slider.Tip({ // step 2
4. getText: function(thumb){ // step 3
5. return String.format('{0}%', thumb.value);
6. }
7. });
```

En el paso uno iniciamos los QuickTip, es importante que esto solo se tiene que declarar una vez en todo el proyecto.

En el paso dos creamos una instancia de “Ext.slider.Tips” y en el paso tres lo que hacemos es sobre escribir el método “getText” esto es con el fin de que en el tip se vea el signo “%” de lo contrario solo se desplegará el valor del slider.

Ya que tenemos nuestros sliders lo siguiente es colocarlos en un panel para poder mostrarlos, el panel quedaría de la siguiente manera:

```
1. new Ext.Panel({
2. renderTo : "tools",
3. title : "Settings",
4. collapsible : true,
5. labelWidth : 60,
6. layout : "form",
7. frame : true,
8. items : [size,opacity]
9. });
```

Con esto el resultado debe ser parecido a esto:



Sliders Dentro del Panel

### Interactuando con la Imagen

Ya que tenemos listo nuestros slider lo siguiente es crear las funciones que harán que nuestra imagen pueda cambiar de tamaño y el valor de la opacidad.

```
1. size.on("change",function(slider,value){
2. el.setWidth(value+"%")
3. el.center(frame);
4. });
```

Lo que el código anterior hace es usar el evento “change” el cual se ejecuta cuando el usuario cambia el valor del slider, recibe como parámetros el slider mismo y el nuevo valor, por ultimo usamos nuevamente “setWidth()” y le pasamos como parámetro el valor que se obtiene del slider, con esto podemos cambiar el tamaño de la imagen, es importante mencionar que no es necesario cambiar la altura de la foto ya que el navegador la calcula basándose el ancho de esta.

Por ultimo solo centramos la foto en el div para que después del cambio de tamaño, siempre este en el centro, recuerda que le pasamos “frame” para que lo centre en el div contenedor.

Lo siguiente es cambiar la opacidad de la foto y eso lo haremos de una manera similar :

```
1. opacity.on("change",function(slider,value){
2. el.setOpacity(value/100);
3. });
```

Básicamente es lo mismo que hicimos para cambiar el tamaño la única diferencia es que usamos “setOpacity()” y le damos el valor del slider, notemos que ese valor es dividido entre cien esto es porque la opacidad es manejada de 0 a 1 por lo cual .5 sería el valor 50 del slider.

## Conclusión

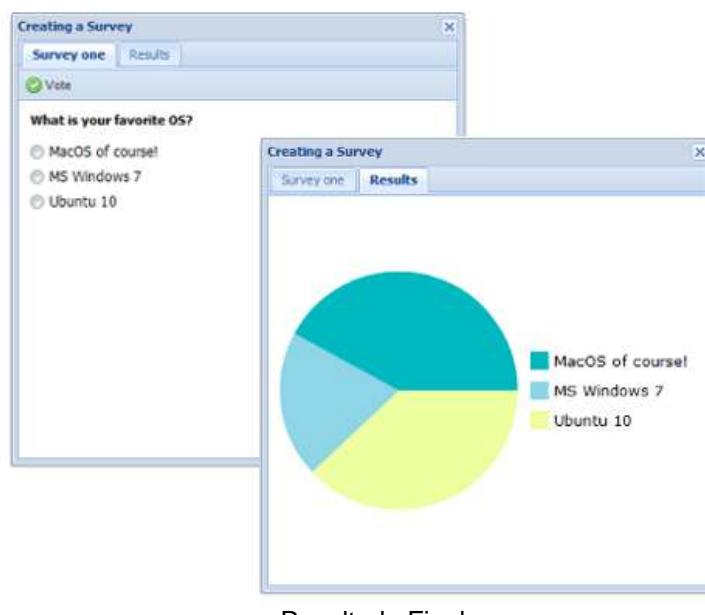
En esta ocasión vimos cómo crear sliders, y darles un uso básico. Estos son muy útiles cuando deseamos hacer cambios a ciertos elementos de nuestro sistema, modificar rangos, etc.

## Una encuesta dinámica

Muchas veces nos interesa saber la opinión de los usuarios, en esta ocasión estudiaremos como crear una encuesta. La información será desplegada vía Ajax, además de tener nuestra encuesta contaremos con una grafica que mostrara los resultados.

El ejemplo que haremos el día de hoy será completamente dinámico, las preguntas, respuestas y resultados de la encuesta estarán almacenadas en una base de datos MySQL y generaremos la interface de acuerdo a lo que se tenga en la base de datos.

Esta es una muestra de lo que se obtendrá al final de este tutorial. Recuerda que puedes descargar el código fuente si te es necesario.



Resultado Final

## La base de datos

Las preguntas y respuestas estarán en dos tablas, una de ellas se llamará “Questions” donde estarán las preguntas y “Answers” donde estarán las respuestas, estas tablas tendrán una relación de “one-to-many” (uno-a-muchos), donde una pregunta puede tener muchas respuestas. El código para generar las tablas se lista a continuación.

1. -- phpMyAdmin SQL Dump
2. -- version 3.2.5
3. -- http://www.phpmyadmin.net
4. --
5. -- Host: localhost
6. -- Generation Time: Sep 29, 2010 at 10:42 AM
7. -- Server version: 5.1.44
8. -- PHP Version: 5.3.2
- 9.
10. SET SQL\_MODE="NO\_AUTO\_VALUE\_ON\_ZERO";
- 11.
12. --
13. -- Database: `test`
14. --
- 15.
16. -----

```

17.
18. --
19. -- Table structure for table `answers`
20. --
21.
22. CREATE TABLE `answers` (
23. `id` int(11) NOT NULL AUTO_INCREMENT,
24. `id_question` int(11) NOT NULL,
25. `answer` varchar(255) NOT NULL,
26. `results` int(11) NOT NULL,
27. PRIMARY KEY (`id`),
28. KEY `id_question` (`id_question`)
29.) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=10 ;
30.
31. --
32. -- Dumping data for table `answers`
33. --
34.
35. INSERT INTO `answers` VALUES(1, 1, 'MacOS of course!', 20);
36. INSERT INTO `answers` VALUES(2, 1, 'MS Windows 7', 10);
37. INSERT INTO `answers` VALUES(3, 1, 'Ubuntu 10', 18);
38. INSERT INTO `answers` VALUES(4, 2, 'Black', 0);
39. INSERT INTO `answers` VALUES(5, 2, 'White', 0);
40. INSERT INTO `answers` VALUES(8, 2, 'Blue', 0);
41. INSERT INTO `answers` VALUES(9, 2, 'Red', 0);
42.
43. -----
44. --
45. --
46. -- Table structure for table `questions`
47. --
48.
49. CREATE TABLE `questions` (
50. `id` int(11) NOT NULL AUTO_INCREMENT,
51. `question` varchar(255) NOT NULL,
52. PRIMARY KEY (`id`)
53.) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=3 ;
54.
55. --
56. -- Dumping data for table `questions`
57. --
58.
59. INSERT INTO `questions` VALUES(1, 'What is your favorite OS?');
60. INSERT INTO `questions` VALUES(2, 'What is your favorite color?');
61.
62. --
63. -- Constraints for dumped tables
64. --
65.
66. --
67. -- Constraints for table `answers`
68. --
69. ALTER TABLE `answers`
70. ADD CONSTRAINT `answers_ibfk_1` FOREIGN KEY (`id_question`) REFERENCES `questions`
 (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

```

La base de datos la he llamado “test”, pero puedes usar el nombre que gustes, solamente cuando se haga la conexión mediante PHP recuerda ponerle el nombre correcto.

## Exponer la información

La información es algo muy importante en una aplicación en ocasiones anteriores hemos usado arreglos, en esta ocasión la obtendremos de una base de datos, por lo tanto necesitamos hacer la conexión vía PHP y un query para sacar la pregunta y sus respuestas de la siguiente manera:

```
1. <?php
2.
3. //step 1
4. $connection=mysql_connect("localhost","root","root") or die("Connection Failed".mysql_error());
5. mysql_select_db("test",$connection)or die("Error loading the DataBase".mysql_error());
6.
7. $survey = $_POST["survey"]; //step 2
8.
9. $query = sprintf("SELECT Q.id, A.id AS id_answer, Q.question, A.answer FROM questions Q, an
swers A
10. WHERE Q.id = A.id_question AND Q.id =%d",mysql_real_escape_string($survey)); //
step 3
11.
12. $result = mysql_query($query);
13.
14. $answers = array();
15.
16. while($row = mysql_fetch_array($result)){ //step 4
17. array_push($answers,array(
18. "idAnswer" => $row["id_answer"],
19. "text" =>$row["answer"]
20.));
21.
22. $question = $row["question"];
23. $idQuestion = $row["id"];
24.
25. }
26.
27. echo json_encode(array(//step 5
28. "success" => true,
29. "question" => $question,
30. "idQuestion" => $idQuestion,
31. "answers" =>$answers
32.));
```

En el paso uno se realiza la conexión a la base de datos, recuerda que debes poner las credenciales adecuadas así como la base de datos que usarás, en mi caso es “test”.

En el paso dos se recibe el parámetro “survey”, este parámetro es el “id” de la pregunta a exponer en formato JSON.

En el paso tres se crea el query que regresa la pregunta con sus respuestas, es una consulta muy sencilla a dos tablas que filtra por el “id\_question” para traer la encuesta adecuada.

En el paso cuatro se itera el resultset que regresó la consulta, dentro del ciclo creamos un arreglo de respuestas y en una variable se asigna la pregunta y su identificador.

En el paso cinco se imprime la información en formato JSON, la respuesta será como el siguiente ejemplo:

1. {"success":true,"question":"Whats your favorite OS?","idQuestion":"1","answers":[{"idAnswer":"1","text":"MacOS of course!"},{"idAnswer":"2","text":"MS Windows 7"}, {"idAnswer":"3","text":"Ubuntu 10"}]}

Con eso es suficiente para poder generar mediante JavaScript la interface de la encuesta ya que tenemos toda la información necesaria.

### Encapsulando el Código

Es necesario encapsular nuestro código para evitar problemas en el futuro, así que crearemos un objeto donde alojaremos el código del tutorial.

```
1. Ext.ns("com.quizzpot.tutorial");
2. com.quizzpot.tutorial.SurveyTutorial= {
3. idQuestion : 1, //step 1
4.
5. init : function(){
6. //initial code goes here
7. }
8. }
9. //step 2
10. Ext.onReady(com.quizzpot.tutorial.CreadoEncuesta.init,com.quizzpot.tutorial. CreadoEncuesta);
```

En el paso uno se ha definido la propiedad “idQuestion”, por defecto será 1, pero podemos cambiarlo para mostrar otra pregunta con sus respectivas respuestas.

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### Solicitando la Información desde el Servidor vía Ajax

Tenemos que solicitar la información al servidor vía Ajax, en nuestro método “init” haremos la solicitud de la siguiente manera:

```
1. Ext.Ajax.request({
2. url : "serverside/questions.php",
3. scope : this,
4. params : {survey:this.idQuestion},
5. success : this.createLayout
6. });
```

Este código ya es conocido solo haremos notar el atributo “params” como se mencionó hace unos momentos este es el parámetro que define la encuesta que desplegaremos, y debe ser un “id” válido de la tabla “Questions”.

Otro punto importante es que no estamos especificando el método para la petición Ajax (POST, GET...) ya que usaremos el método que trae por defecto (POST).

Con “success” indicamos la función que procesará la respuesta si todo ha salido bien, además, si definimos la configuración “scope” la función se ejecutará en el contexto que le indiquemos, en este caso “this” que hace referencia al objeto sobre el que estamos trabajando, esto es algo muy importante.

Como hemos solicitado el idQuestion igual a uno, obtenemos el siguiente resultado:

```
1. {"success":true,"question":"Whats your favorite OS?","idQuestion":"1","answers":[{"idAnswer":"1","text":"MacOS of course!"},{"idAnswer":"2","text":"MS Windows 7"}, {"idAnswer":"3","text":"Ubuntu 0"}]}
```

El resultado puede variar dependiendo de cual es la información que tenemos en nuestra base de datos y del idQuestion que solicitemos.

### Creando el Layout

La función “createLayout” procesará la respuesta del servidor, primero obtendremos el JSON y lo convertiremos a objetos en memoria para poder usarlos de una manera sencilla. Creamos la ventana y los tabs que contendrán la información.

```

1. createLayout : function(response,options){
2. var survey = Ext.decode(response.responseText); //step 1
3.
4. this.tabs = new Ext.TabPanel({ //step 2
5. border : false,
6. activeTab : 0,
7. items : [this.createForm(survey),this.createChar()] //step 3
8. });
9. var win = new Ext.Window({
10. title : "Creating a Survey",
11. width : 400,
12. height : 400,
13. layout : "fit",
14. items : [this.tabs],
15. });
16. win.show();
17. }

```

En el paso uno obtenemos la información de la respuesta Ajax y la colocamos en la variable “survey” para después poder acceder a ella y poder usarla, esto lo hacemos con el método “Ext.decode”.

En el paso dos solo creamos las pestañas que mostrarán la encuesta y los resultados, nótese que se ha creado una propiedad “this.tabs”, esto es para que podamos usar los tabs mas adelante, inclusive en otros métodos del mismo objeto.

En el paso tres se ejecuta la función “createForm” y “createChar”, estos métodos se encargarán de crear los componentes adecuados, de esta manera estamos separando las tareas de manera más específica.

Por ahora puedes comentar o borrar el método “createChar” ya que estaremos hablando del primer método, mas adelante regresaremos aquí a explicar el segundo método.

### **Creando la Encuesta**

En este momento crearemos una de las funciones que se encargará de mostrar la encuesta. Lo primero que haremos es procesar la información que se recibió en la petición a Ajax.

```

1. createForm : function(survey){
2. var radios = [];
3.
4. Ext.each(survey.answers,function(answer){
5. radios.push({boxLabel: answer.text, name: "idAnswer", inputValue: answer.idAnswer});
6. });
7.
8. //.... Seguiremos escribiendo aquí
9. }

```

El arreglo “radios” servirá para crear dinámicamente los “RadioButtons”, crearemos uno por cada posible respuesta, de esta manera desplegaremos dinámicamente todas las opciones que vienen de la base de datos, nótese que el “name” está definido como “idAnswer” y es el mismo para todos, además de que el “inputValue” contiene el “idAnswer” de la base de datos, la propiedad “boxLabel” simplemente muestra el texto que aparece al lado del radio, en este caso la respuesta a seleccionar.

Lo siguiente es crear el grupo de radios, de la siguiente manera:

```

1. var options = new Ext.form.RadioGroup({
2. hideLabel : true,
3. fieldLabel : "Survey",
4. columns : 1,
5. items : [radios]
6. });

```

Es muy importante mencionar que todo este código aun es dentro de la función `createForm`. El atributo “`hideLabel`” esconde el “`fieldLabel`” y con esto podemos darle una mejor vista a nuestra encuesta.

Además del grupo de radios que contienen las respuestas a la encuesta necesitamos desplegar la pregunta principal y también almacenar los identificadores para poder guardar los resultados más adelante.

```
1. //Código previo...
2. var question = { //step 1
3. xtype : "panel",
4. border : false,
5. bodyStyle : "padding-bottom:10px;",
6. html : "<h1>" + survey.question + "</h1>"
7. };
8. var idQuestion = { //step 2
9. xtype : "hidden",
10. name : "idQuestion",
11. value : survey.idQuestion
12. };
13.
14. this.form = new Ext.FormPanel({ //step 3
15. title : "Survey one",
16. url : "serverside/submit.php",
17. bodyStyle : "padding: 10px;",
18. border : false,
19. tbar : [{text: "Vote", iconCls:"vote-icon" , scope: this}], //step 4
20. items : [question,options,idQuestion]
21. });
22.
23. return this.form;
```

En el paso uno se crea la pregunta que desplegaremos, es un panel que únicamente contiene el texto de la pregunta que viene de la base de datos.

En el paso dos se crea un campo de tipo “`hidden`”, este campo almacena el id de la encuesta, es necesario para que más adelante cuando guardemos los resultados sepamos que registro modificar en la base de datos.

En el paso tres se crea el formulario que contiene los componentes anteriores, aquí estamos definiendo la propiedad “`url`” que es a donde se enviarán los resultados al hacer un “`submit`” al formulario.

En el paso cuatro se crea el botón “votar” en la barra de herramienta, es importante mencionar que se esta colocando una imagen al botón, para esto es necesario definir una clase CSS (`vote-icon`) en nuestra hoja de estilos o documento HTML de la siguiente manera.

```
1. .vote-icon{background:transparent url(accept.png) 0 0 no-repeat !important;}
```

Hasta ahora hemos creado el formulario que despliega la pregunta y sus respuestas de la encuesta, la siguiente imagen muestra los resultados que tenemos.



Encuesta

Solo hemos desplegado la información que hemos obtenido desde nuestra base de datos. Lo siguiente es guardar el voto del usuario una vez que seleccione una respuesta y de click sobre el botón “Vote”.

### Guardando en nuestro Servidor

Lo primero que haremos será agregar un handler al botón “Vote”, de esta manera podemos agregarle acciones para cuando el usuario de click sobre este.

```

1. this.form = new Ext.FormPanel({
2. title : "Survey one",
3. url : "serverside/submit.php",
4. bodyStyle : "padding: 10px;",
5. border : false,
6. tbar : [{text: "Vote", iconCls:"vote-icon",handler: this.sendData, scope: this}], //step 1
7. items : [question,options,idQuestion]
8. });

```

En el paso uno agregamos el “handler” al botón, esta función se dispara cada vez que se da click sobre botón. En este ejemplo el “handler” tiene definida la función “sendData” la cual se encarga de hacer la petición Ajax al servidor enviando la respuesta seleccionada por el usuario

```

1. sendData : function(){
2. var id = this.form.getForm().getValues().idAnswer; //step 1
3.
4. if(!Ext.isEmpty(id)){ //step 2
5. this.form.getForm().submit({
6. scope : this,
7. success : this.showResults,
8. failure : function(response){
9. console.debug(response);
10. }
11. });
12. }else{
13. Ext.Msg.alert("Error","Please select an answer before sending your vote!");
14. }
15. },

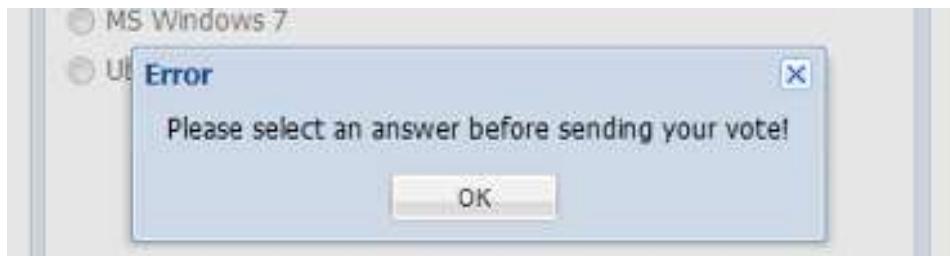
```

El paso uno es muy interesante, en ExtJS por medio de el BasicForm podemos acceder al objeto que maneja la información del formulario, usando el método “getValues” podemos tomar los valores que se han capturado. En este caso a la variable “id” le asignamos el “idAnswer” de la respuesta seleccionada.

En el paso dos revisamos que el usuario ha seleccionado alguna respuesta, si la variable “id” trae algún valor es porque el usuario si se seleccionó alguna respuesta antes de dar click sobre el botón “votar”, de ser así hacemos el submit al formulario.

Es importante mencionar que el atributo “success” se emplea otra función que es showResults, lo que hace esta función es simplemente mandar un mensaje de alerta informándonos que todo salió bien, aquí también usamos la configuración “scope”, esto permite definir el contexto donde se ejecutará la función asignada al callback “success” y “failure”.

Si la variable “id” se encuentra vacía solamente mandamos un mensaje de alerta diciendo que tiene que seleccionar una respuesta antes de poder votar.



Mensaje de Error

### Actualizando los resultados en la base de datos

En los pasos anteriores realizamos el “submit” al formulario una vez que el usuario da click sobre el botón “Vote”, ahora vamos a actualizar los resultados, incrementando en uno el campo “results” de la respuesta seleccionada.

En el archivo “serevrside/submit.php” escribimos el siguiente código.

```
1. <?php
2. //step 1
3. $conección=mysql_connect("localhost","root","");
4. or die("no se puede conectar".mysql_error());
5. mysql_select_db("test",$conección)or die("error en la selección de la base".mysql_error());
6. //atep 2
7. $idAnswer = $_POST['idAnswer'];
8. $idQuestion = $_POST['idQuestion'];
9.
10. //step 3
11. $query = sprintf("SELECT results FROM answers WHERE id=%d and id_question=%d",
12. mysql_real_escape_string($idAnswer),
13. mysql_real_escape_string($idQuestion));
14.
15. $result=mysql_query($query);
16.
17. while($row=mysql_fetch_array($result)){
18. $votes= $row["results"];
19. }
20.
21. //step 5
22. $votes = $votes + 1;
23.
24. $query = sprintf("UPDATE answers SET results = %d WHERE id=%d and id_question=%d",
25. $votes,
26. mysql_real_escape_string($idAnswer),
27. mysql_real_escape_string($idQuestion));
28.
29. $rs = mysql_query($query);
30. echo json_encode(array(
31. "success" => true
32.));
```

En el paso uno realizamos la conexión a la base de datos, asegúrate de que poner las credenciales correctas.

En el paso dos recibimos vía POST los parámetros que nos indican la pregunta y respuesta seleccionada, estos parámetros se enviaron automáticamente al hacer "submit" al formulario que definimos en ExtJS.

En el paso tres solicitamos a la base de datos la respuesta elegida por el usuario, nótense el uso de la función "mysql\_real\_escape\_string" para evitar algún ataque de SQL Injection.

En el paso cinco incrementamos en uno el valor actual que está guardado en la base de datos y luego hacemos la actualización a la tabla "Answers".

### Mensaje de éxito

Una vez que se ha guardado el voto crearemos la función "showResult" que es el callback para cuando la comunicación ha sido satisfactoria, dentro de esta función solamente mandaremos un mensaje indicándole al usuario que su voto ha sido guardado.

```
1. showResults : function(response){
2. Ext.Msg.alert("Alert","Thanks you for your vote!");
3. }
```

Esta función se crea dentro del objeto principal.



En este momento hemos hecho lo necesario para que la encuesta se cargue desde nuestro servidor, se muestre al usuario, que este pueda seleccionar una respuesta a la pregunta y guardar su respuesta, todo esto de una manera muy sencilla. Pero también es importante mostrar las respuestas de otros usuarios, para esto crearemos una grafica donde se mostraran los resultados de la encuesta.

### Gráfica con Resultados

Lo primero que haremos es crear la consulta a la base de datos, de la cual obtendremos la información que formara la grafica. En el archivo "serverside/information.php" escribiremos el siguiente código.

```
1. <?php
2. $coneccion = mysql_connect("localhost","root","");
3. or die("no se puede conectar".mysql_error());
4.
5. mysql_select_db("test",$coneccion)or die("error en la seleccion de la base".mysql_error());
6.
7. $idQuestion = $_POST["idQuestion"];
8.
9. $query = sprintf("SELECT Q.question, A.answer, A.results FROM questions Q, answers A
10. WHERE Q.id = A.id_question AND Q.id = %d",mysql_real_escape_string($idQuestion));
11. $result = mysql_query($query);
12.
13. $info = array();
14. while($row = mysql_fetch_array($result)){
15. array_push($info,array(
16. "answer" => $row["answer"],
17. "votes" => $row["results"]
18.));
19. $question = $row["question"];
20. }
```

```

20. }
21. echo json_encode(array(
22. "success" => true,
23. "question" => $question,
24. "information" => $info
25.));

```

En el código anterior se consultan los resultados de una encuesta la cual es definida mediante el parámetro “idQuestion”, la información es regresada en formato JSON de la siguiente manera:

```

1. {"success":true,"question":"What is your favorite OS?","information":[{"answer":"MacOS of course!",
 "votes":20},{"answer":"MS Windows 7","votes":10},{"answer":"Ubuntu 10","votes":18}]}

```

Como se puede apreciar, los resultados vienen dentro del arreglo “information”, con este formato en cuenta podemos crear la gráfica de una manera muy sencilla.

Si lo notaste, cuando creamos el layout invocamos una función que se llamaba “createChar” dentro del arreglo ítems del TabPanel, el objetivo de esta función es crear la gráfica de la siguiente manera.

```

1. createChar : function(){
2. //step 1
3. this.storeChar = new Ext.data.JsonStore({
4. url : "serverside/information.php",
5. fields : [{name:"answer"},{name:"votes", type:"float"}],
6. root : "information"
7. });
8.
9. //step 2
10. var pieChart = new Ext.chart.PieChart({
11. store : this.storeChar,
12. dataField : "votes",
13. categoryField : "answer",
14. extraStyle : {
15. legend : {
16. display : 'right',
17. padding : 5,
18. font : {
19. family: 'Tahoma',
20. size: 13
21. }
22. }
23. }
24. });
25.
26. //step 3
27. var results = new Ext.Panel({
28. title : "Results",
29. items : [pieChart],
30. listeners: {
31. scope : this,
32. show : this.refresh
33. }
34. });
35.
36. return results;
37. },

```

En el paso uno creamos el store que consulta la información en el servidor, definimos la URL, el root y los campos que tendrá cada registro, nótese que no estamos cargando el store, solamente lo estamos definiendo.

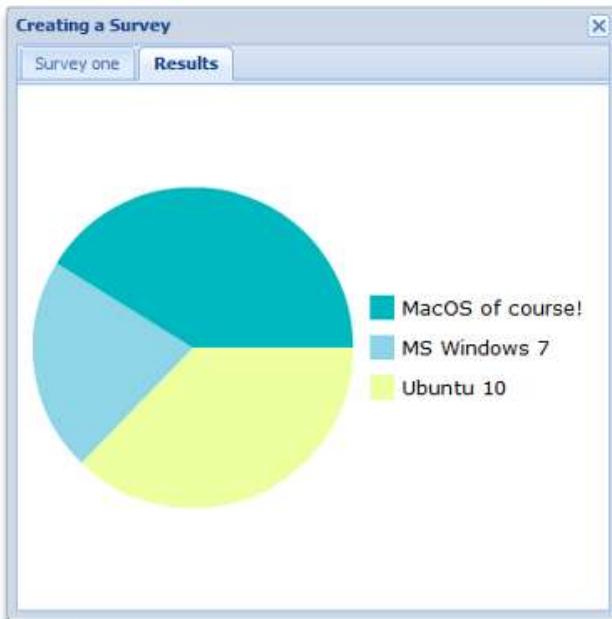
En el paso dos creamos la gráfica de pastel o PieChart, le asignamos el store que recién hemos creado "this.storeChar" y definimos la propiedad "dataField", con esto le indicamos al componente el campo en el store que contendrá el número de votos, este campo es muy importante para que se pueda dibujar correctamente la gráfica, la propiedad "categoryField" también es indispensable, esta configuración indica las agrupaciones que hará la gráfica para mostrarse, el capo "answer" contiene el texto de la respuesta.

Para agregar estilos a la gráfica usamos la configuración "extraStyle", y al definir la propiedad "display" podemos posicionar el lugar donde queremos desplegar las leyendas de las categorías, en este caso en la parte derecha (right) pero podríamos posicionarla en algún otro lugar como el "bottom, top o left", las otras configuraciones que usamos se explican así mismas.

En el paso tres creamos el panel que contendrá la gráfica y que irá dentro del TabPanel principal, aquí estamos agregándole un listener al evento "show" para que en ese momento se ejecute la función "refresh" cuyo objetivo es solicitar la información que contendrá la gráfica.

1. refresh : function(){
2.     this.storeChar.load({params:{idQuestion:this.idQuestion}});
3. }

Con esto es suficiente para poder apreciar la gráfica de los resultados, actualiza tu navegador y podrás ver algo como la siguiente imagen.



Grafica de Resultados

### Conclusiones

En este tutorial se han creado componentes de manera dinámica, esta misma técnica se puede usar para crear otro tipo de componente dinámicamente por ejemplo botones en un toolbar, grids, ventanas, y todo lo que se nos ocurra.

## Integración del TreePanel y TabPanel

Es muy común que en alguna aplicación usemos un layout de tipo “border”, generalmente en la región “oeste” colocamos el menú principal usando un “TreePanel” para desplegar las secciones o categorías y en la región central desplegamos los módulos de nuestra aplicación.

En esta ocasión veremos cómo desplegar múltiples contenidos que se tienen en un TreePanel, utilizando un TabPanel para visualizar el contenido del árbol.

Esta es una muestra de lo que se obtendrá al final de este tutorial. Recuerda que puedes descargar el código fuente si es necesario.



### Encapsulando el código

Es muy necesario encapsular nuestro código para evitar problemas en el futuro, así que crearemos un objeto donde alojaremos el código del tutorial.

1. Ext.ns("com.quizzpot.tutorial");
- 2.
3. com.quizzpot.tutorial.IntegracionTutorial = {
4. init: function(){
5. //el código irá aquí
6. }
7. }
8. Ext.onReady(com.quizzpot.tutorial. IntegracionTutorial.init,com.quizzpot.tutorial. IntegracionTutorial);

La función “init” se ejecutará tan pronto como el DOM esté listo, por lo tanto ahí debemos colocar el código que necesitamos ejecutar primero.

### Creando el Layout

En primer lugar vamos a crear el layout que tendrá nuestra aplicación, en la parte izquierda necesitamos que aparezca el TreePanel y en la parte derecha el panel donde se verá la información al dar clic sobre alguna opción del menu.

Como hemos visto en tutoriales pasados cuando usamos un layout de tipo “border” es necesario que definamos las regiones que contendrá.

1. var center = {
2. xtype : "panel",

```

3. region : "center",
4. layout : "fit",
5. border : false,
6. margins : {bottom:3,right:3}
7. };
8.
9. var west = {
10. xtype : "panel",
11. region : "west",
12. width : 150,
13. split : true,
14. collapsible: true,
15. title : "West region",
16. margins : "0 0 3 3"
17. };
18.
19. var north = {
20. xtype : "panel",
21. region : "north",
22. height : 50,
23. html : "North region",
24. margins : {top:3,bottom:3,left:3,right:3}
25. };
26.
27. var main = new Ext.Panel({
28. renderTo : "content",
29. layout : "border",
30. height : 600,
31. items : [center,west,north]
32. });

```

No entraremos en detalles del layout ya que esto se ha tratado en tutoriales pasados solo comentar que la propiedad “renderTo” tiene asignado “content” el cual es el ID de un div definido en el DOM.

Ya que tenemos las regiones definidas este es el resultado.



### Creación del menú principal

Ya que tenemos nuestro layout listo podemos introducir los componentes. Primero crearemos el TreePanel. Este es un proceso que se realizado ya en varios tutoriales previos.

```

1. var tree= new Ext.tree.TreePanel({
2. border : false,
3. autoScroll : true,
4. root : this.getData()
5. });

```

Si lo notaron, hay una llamada al método “this.getData”, este método está definido dentro del objeto principal de la siguiente manera:

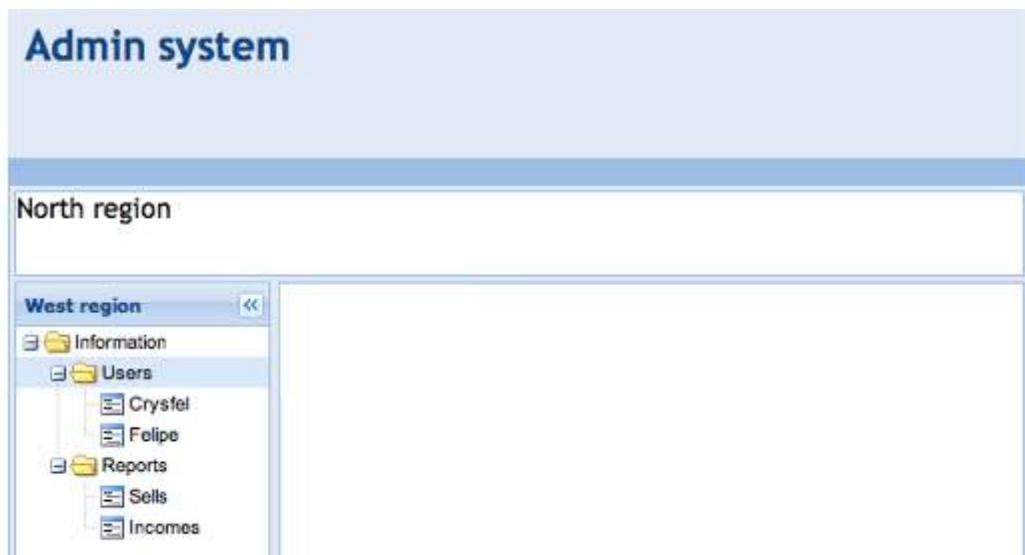
```
1. getData : function(){
2. var root = {
3. text:'Informacion',
4. expanded: true,
5. children:[
6. {
7. text:'Usuarios',
8. children:[{text:'Felipe',leaf:true},
9. {text:'Omar',leaf:true}]
10. },{
11. text:'Departamentos',
12. children:[{text:'Compras',leaf:true},
13. {text:'Ventas',leaf:true}]
14. }
15.]
16. }
17. return root;
18. }
```

Este método tiene por objetivo de generar los nodos que tendrá el árbol.

Por último es necesario asignar el TreePanel que recién creamos a la región oeste (west) de nuestro layout principal que creamos al inicio del tutorial.

```
1. var west = {
2. xtype : "panel",
3. region : "west",
4. width : 150,
5. split : true,
6. collapsible:true,
7. title : "West region",
8. margins : "0 0 3 3",
9. items : [tree] //asignnames el tree en la parte oeste..
10. };
```

Hasta este punto deberíamos tener algo semejante a la siguiente pantalla:



Colocando el Árbol

## Pestaña inicial

Ya que tenemos nuestro árbol creado procederemos a crear la pestaña Home, la cual será una pestaña fija ya que es aquí donde podemos poner la información inicial que verá el usuario.

view plaincopy to clipboardprint?

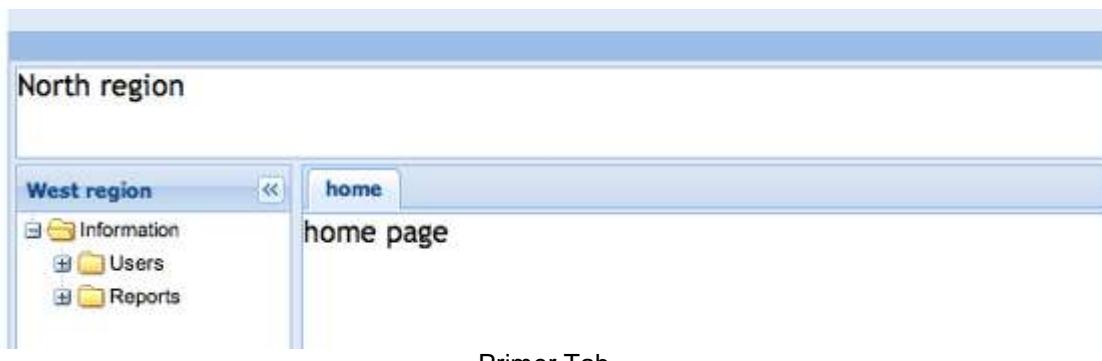
```
1. //creacion de el primer tab (home)
2. var home= new Ext.Panel({ //step 1
3. title : "Home",
4. html : "home page"
5. });
6.
7. //Creamos el TabPanel para almacenar los tabs
8. this.tabs = new Ext.TabPanel({ //step 2
9. border : false,
10. activeTab : 0,
11. enableTabScroll : true, //hacemos que sean recorridas
12. items : [home]
13. });
```

En el paso uno creamos la pestaña con sus propiedades, aquí podríamos poner cualquier cosa que necesitemos, para dejar las cosas más sencillas solamente le he definido un texto.

En el paso dos creamos el panel en el cual se alojaran las pestañas que serán creadas dinámicamente cuando el usuario de click sobre algún elemento del menú principal. Por el momento solo cuenta con la pestana Home.

Una vez que tenemos nuestro TabPanel listo lo siguiente es ponerlo en el lugar correcto, lo haremos en la región central ya que es aquí donde normalmente va el contenido principal.

```
1. var center = {
2. xtype : "panel",
3. region : "center",
4. layout : "fit",
5. border : false,
6. margins : {bottom:3,right:3},
7. items : [
8. xtype : "panel",
9. items:[this.tabs] //Asignamos el tabpanel a la región central
10.]
11.};
```



## Asignar eventos al menú principal

Hasta ahora solo contamos con nuestro layout terminado, el TreePanel y el TabPnel. Es hora de hacer de nuestra aplicación algo más dinámica, para esto haremos que cuando el usuario de click sobre un nodo del árbol se abra una pestaña con la información sobre el nodo al cual se le dio click.

```
1. tree.on('click',function(node){
2. this.addTab(node);// esta función será creada a continuación
3. },this);
```

En el código anterior asignamos un listener al evento “click” del TreePanel, y con esto cuando el usuario de click sobre uno de los nodos esta función se ejecutara y se le dará como parámetro el “nodo” en el cual se ha dado el click.

Lo siguiente es crear una pestaña y agregarla al TabPanel. En esta pestaña es donde se visualizara la información que contiene el nodo en el cual se ha dado click, todo esto lo hará la función “addTab” que es ejecutada dentro del listener.

```
1. addTab : function(node){
2. var tab = this.tabs.findById("id-"+node.id); // step 1
3.
4. if(!tab){ //step 2
5. tab = new Ext.Panel({
6. id : "id-"+node.id,
7. closable: true, //para que se pueda cerrar el tab
8. title : node.attributes.text, //step 3
9. html : 'This is the main content'
10. });
11. this.tabs.add(tab); //step 4
12. this.tabs.doLayout();
13. }
14.
15. this.tabs.activate(tab); //step 5
16. }
```

En el paso uno busca si el tab ya ha sido creado anteriormente, esta búsqueda la hace mediante el id.

En el paso dos valida si se ha encontrado el tab, y si no existe procedemos a crearlo, la idea de esto es que solamente sea creado una sola vez.

En el paso tres observemos que es colocado el título al tab, lo interesante es que bajo “atributo” encontramos toda la información que contiene el nodo, y con esto accedemos a su texto que usaremos como titulo de la pestaña.

En el paso cuatro agregamos el tab que acabamos de crear al TabPanel y lo refrescamos para que pueda mostrar en pantalla las modificaciones que ha sufrido.

En el paso cinco activamos el tab del nodo en el cual se ha dado click, si ya existía solamente se hace visible.

## Conclusiones

El tutorial del día de hoy es una mezcla de tutoriales pasados, lo interesante aquí es la manera en que se integran para crear una aplicación sencilla.

## ExtJS Desktop – Creación de módulos

En este video-tutorial se muestra como crear módulos dentro del Desktop de ExtJS para que comiences a desarrollar tus aplicaciones de manera rápida y utilizando las bondades de Desktop que provee la librería.

### Iniciar el Desktop

El primer paso es copiar los archivos necesarios para poder trabajar con el Desktop, debes copiar la carpeta que se encuentra dentro del directorio `ext-3.2.1/examples/desktop` al directorio raíz o en cualquier otro lugar donde quieras almacenar tu aplicación.

### Modificar las referencias

Una vez que tenemos la carpeta desktop en el directorio raíz es necesario abrir el archivo “`desktop.html`” y cambiar las referencias a la librería de extjs.

1. `<link rel="stylesheet" type="text/css" href="ext-3.2.1/resources/css/ext-all.css" />`
- 2.
3. `<script type="text/javascript" src="ext-3.2.1/adapter/ext/ext-base.js"></script>`
4. `<script type="text/javascript" src="ext-3.2.1/ext-all-debug.js"></script>`

### Importar nuestro módulo

Cuando trabajamos con el Desktop (también cuando no lo hacemos) es recomendable escribir nuestro código javascript dentro de objetos para que esté encapsulado y evitar problemas a futuro, para este ejemplo voy a importar una extensión que es un cliente Twitter creado con Extjs (yo la desarrolle para el curso de extjs), así que los archivos a importar serán los siguientes.

1. `<link rel="stylesheet" type="text/css" href="js/Twittext/css/style.css" />`
- 2.
3. `<script type="text/javascript" src="js/Twittext/TwittextUI.js"></script>`
4. `<script type="text/javascript" src="js/Twittext/Twittext.js"></script>`

### Agregando nuestro módulo al Desktop

Una vez que hemos importado al documento html nuestro módulo necesitamos inicializarlo y agregar un link dentro del menú “start” del desktop, para eso podemos editar el archivo “`sample.js`” o bien crear otro archivo que contenga el siguiente código.

```
1. MyDesktop.Twittext= Ext.extend(Ext.app.Module, {
2. id:'twittext-id',
3.
4. init : function(){
5. this.launcher = {
6. text: 'Twittext Client',
7. iconCls:'grid-win',
8. handler : this.createWindow,
9. scope: this
10. }
11. },
12.
13. createWindow : function(){
14. var desktop = this.app.getDesktop();
15. var win = desktop.getWindow(this.id);
16. if(!win){
17. var twittext = new Ext.ux.Twittext({
18. url : "serverside/App.php",
19. border : false
20. });
21. win = desktop.createWindow({
22. title: 'Twittext Client'
23. })
24. }
25. return win;
26. }
27.}, {
28. name: 'twittext',
29. icon: 'twittext/icon.png',
30. title: 'Twittext Client',
31. width: 400, height: 300
32.});
```

```

23. id: this.id,
24. title:'Twittext Client',
25. width:300,
26. height:500,
27. iconCls: 'grid-win',
28. shim:false,
29. animCollapse:false,
30. border:false,
31. constrainHeader:true,
32.
33. layout: 'fit',
34. items: twittext
35. });
36. }
37. win.show();
38. }
39.);

```

En el código anterior existen dos cosas muy importantes, lo primero es extender de la clase “Ext.app.Module” la cual dispara el método “init” de nuestra clase, dentro del cual se esta agregando el link al menu “start”, el siguiente punto a considerar es la creación del método “createWindow” este método se ejecutará cuando el usuario de click sobre la opción en el menu “start”, así que en este método se debe crear la ventana que contenga nuestro componente o módulo.

### **Regitrando el módulo en el Desktop**

Hasta ahora todavía no podríamos ver nada en nuevo en el desktop, a este punto necesitamos regitrar el nuevo módulo para que pueda ser desplegado, para eso simplemente modificamos el archivo “sample.js” de la siguiente manera.

```

1. MyDesktop = new Ext.app.App({
2. init :function(){
3. Ext.QuickTips.init();
4. },
5.
6. getModules : function(){
7. return [
8. new MyDesktop.GridWindow(),
9. new MyDesktop.TabWindow(),
10. new MyDesktop.AccordionWindow(),
11. new MyDesktop.BogusMenuModule(),
12. new MyDesktop.BogusModule(),
13. new MyDesktop.Twittext() // Creamos una instancia de nuestro módulo
14.];
15. },
16.
17.

```

Dentro del método “getModules” creamos una instancia del componente que definimos en el paso anterior, con esto nuestro módulo estará lito para ser utilizado.