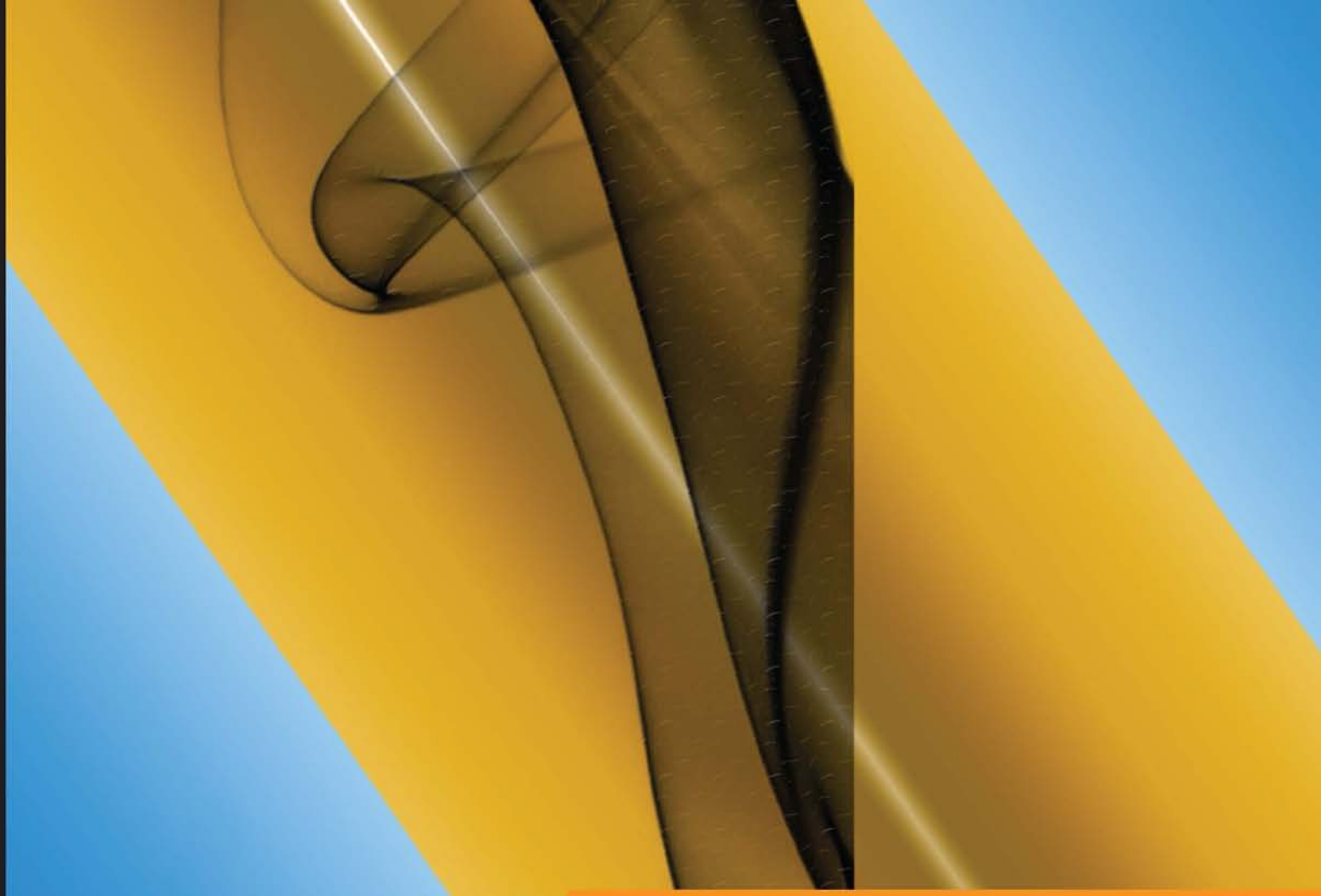


INSTANT

Short | Fast | Focused

Ext JS Starter

Find out what Ext JS actually is, what you can do with it, and why it's so great

Nagarajan Bhava

[PACKT] PUBLISHING

Instant Ext JS Starter

Find out what Ext JS actually is, what you can do with it, and why it's so great

Nagarajan Bhava

BIRMINGHAM - MUMBAI

Instant Ext JS Starter

First published: April 2013

Production Reference: 1220413

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-610-8

www.packtpub.com

Credits

Author

Nagarajan Bhava

Project Coordinator

Suraj Bist

Reviewer

Indroniel Deb Roy

Proofreader

Ting Baker

Acquisition Editor

Usha Iyer

Graphics

Ronak Dhruv

Commissioning Editor

Ameya Sawant

Production Coordinator

Melwyn D'sa

Technical Editor

Lubna Shaikh

Cover Work

Melwyn D'sa

About the Author

Nagarajan Bhava is a User Interface Architect, who actively explores, designs, and develops user interfaces using the latest UI technologies. He has over 14 years of experience—12 years in IT and 2 years in Control system design engineering.

His work on user interfaces started during pre-Ajax days, for a Business Analytic product with thick-client and Windows desktop-like features in a thin web browser client interface and he is continuing his development for SaaS-based Cloud security service.

He primarily works on user interface designs, developing intuitive rich user interfaces using the latest User Interface frameworks (such as jQuery, MooTools, YUI, DOJO, and Ext JS), Ajax, JavaScript, HTML5, CSS, and Web 2.0 technologies, along with Java J2EE backend technologies.

He has recently taken an interest in designing user interfaces for mobile devices.

He is creative and loves to research various upcoming UI trends and implements them wherever applicable. He has a passion for UI design and is an artist by hobby.

I would like to thank my parents, Mohanlal and Gokhila, for the sacrifices they made to bring me so many opportunities.

I would like to thank my wife, Vaishnavi, for giving me the time, space, love, and support needed to work on this project.

I would also like to thank my co-worker, Indroniel Deb Roy, for reviewing the book.

And a special thanks to the editorial staff at Packt Publishing for believing in me.

About the Reviewer

Indroniel Deb Roy works as a Senior UI Engineer. He has worked with various JavaScript technologies, such as ExtJs, Sencha Touch, JQuery, JQuery Mobile, and Backbone for the past four years. He also worked in iOS Objective C, Flex, J2EE server-side technologies, and has recently started with Node.js.

He has worked in companies such as Yahoo, Oracle, and Novell in the past. He is a co-founder of mTgr8 Inc and Netitude Inc.

He has written several technical papers in the field of Ajax, XML and Flex.

You can find his blogs at `http://indronieldebroy.sys-con.com/`.

I would like to thank my family for their support.

---

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt Publishing offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt Publishing books and eBooks.

Table of Contents

Instant Ext JS Starter

Welcome to *Instant Ext JS Starter*. This book has been especially created to provide you with all the information that you need to start with the Ext JS @ Sencha JavaScript framework. You will understand what the framework does, get started with building your first browser-based application/widget, and discover its rich and modern UI components.

This document contains the following sections:

- *So, what is Ext JS?* looks at what Ext JS actually is, what you can do with it, and why it's so great.
- *Installation and environment setup* teaches you how to download and install Ext Js with minimum fuss and then how to set up a quick working environment.
- *Quick Start – Creating and defining your Ext JS application* explains to you the basics of the framework, class system, extend, apply, and a few other EXT JS built-in basic functions.
- *Top features you'll want to know about* explains the basics of the framework, class system, types of components, how to lay out the components and their containers, event mechanisms, and custom components and theming.
- *People and places you should get to know* provides you with many useful links to the project page and forums, as well as a number of helpful articles, tutorials, and contributors.

So, what is Ext JS?

Ext JS is a JavaScript library that makes it (relatively) easy to create cross-platform browser and desktop-style user interfaces in a web application. It supports the following:

- Component model
- Layouts
- Plugin free charting
- Drawing
- CSS theming

JavaScript is a classless, prototype-oriented language but Ext JS follows a class-based approach to make the code extensible and scalable over time. Class names can be grouped into packages with namespaces using the object property dot-notation (.). Namespaces allow developers to write structured and maintainable code, use libraries without the risk of overwriting functions, avoid cluttering the global namespace, and provide an ability to encapsulate the code.

The strength of the framework lies in its component design. The bundled, basic default components can be easily extended as per your needs and the extended components can be re-used. A new component can also be created by combining one or more default components.

The framework includes many default components such as windows, panels, toolbars, drop-down menus, menu bars, dialog boxes, grids, trees, and much more, each with their own configuration properties (configs), component properties, methods, events, and CSS classes.

The **configs** are user-configurable at runtime while instantiating, whereas component properties are references to objects used internally by class. **Component properties** belong to the prototype of the class and affect all the instances of the class. The **properties** of the individual components determine the look and feel. The **methods** help in achieving a certain action. The **user interaction** triggers the equivalent Ext JS events apart from triggering the DOM events.

A cross-browser web application with header, footer, left column section with links, a content with a CSS grid/table (with add, edit, and delete actions for each row of the grid), and a form with few text fields and a submit button can be created with ease using Ext JS's layout mechanism, few default components, and the CSS theme.

For the preceding application, the border layout can be used with the north region for the header, south region for the footer, west region for the left column links, and center region for the content. The content area can have a horizontal layout, with the grid and form panel components with text fields and buttons.

Creating the preceding application from scratch without using the framework will take a lot more time than it would take by using it. Moreover, this is just one screen, and as the development progresses with more and more features, incorporating new layouts and creating new components will be a tedious process.

All the components or a group of components with their layout can be made a custom component and re-used with different data (that is, the grid data can be modified with new data and re-used in a different page).

Developers need not worry about the cross-platform compatibility issues, since the framework takes care of this, and they can concentrate on the core logic.

The helper functions of the `Ext.DomQuery` class can be used for querying the DOM. The error handling can be done by using the `Ext.Error` class, which is a wrapper for the native JavaScript Error object.

A simple webpage with a minimal UI too can make use of this framework in many ways.

Native JavaScript offers utility classes such as `Array`, `Number`, `Date`, `Object`, `Function`, and `String`, but is limited in what can be done with it across different browsers. Ext JS provides its own version of these classes that works in all the browsers along with offering extra functionality.

Any Ext JS component can be added to an existing web page by creating an instance of it. For example, a tab feature can be added to an existing web page by creating a new Ext JS `Ext.tab` tab component and adding it to an existing `div` container, by referring the div elements `id` attribute to the `renderTo` config property of the tab. The backend communication with your server-side code can be done by using simplified cross-browser `Ext.Ajax` class methods.

Ext JS 4 supports all major web browsers, from Internet Explorer 6 to the latest version of Google Chrome. The recommended browsers for development and debugging are Google Chrome 10+, Apple Safari 5+, and Mozilla Firefox 4+.

Both commercial and open source licenses are available for Ext JS.

Installation and environment setup

In five easy steps, you can be ready with Ext JS and start the development.

Step 1 – What do you need?

You need the following components for the installation and environment setup:

- **Web browser**: Any of the leading browsers mentioned in previous section. For this book, we will consider Mozilla Firebug with the Firebug debugger plugin installed.
- **Web server**: To start with, a local web server is not required, but it will be required if communication with a server is required to make AJAX calls.
- **Ext JS 4 SDK**: Download the Ext JS bundle from `http://www.sencha.com/products/extjs/download/`.

Click on the **Download** button on the left side of the page.

Step 2 – Installing the browser and debugger

Any supported browser mentioned in the previous section can be used for the tutorial. For simplicity and debugging options, we will use the latest Firefox and Firebug debugger plugin. Download the latest Firefox plugin from `http://www.mozilla.org/en-US/firefox/fx/#desktop` and Firebug from `https://getfirebug.com/`.

Other browser debugging options are as follows:

- **Google Chrome**: Chrome Developer Tools (**Tools | Developer tools**)
- **Safari**: Go to **Settings | Preferences |Advanced**, select **Show Develop menu in menu bar**; navigate to **Develop | Show Web Inspector**.
- **Internet Explorer**: Go to **Tools | Developer Tools**

Step 3 – Installing the web server

Install the web server and unpack Ext JS.

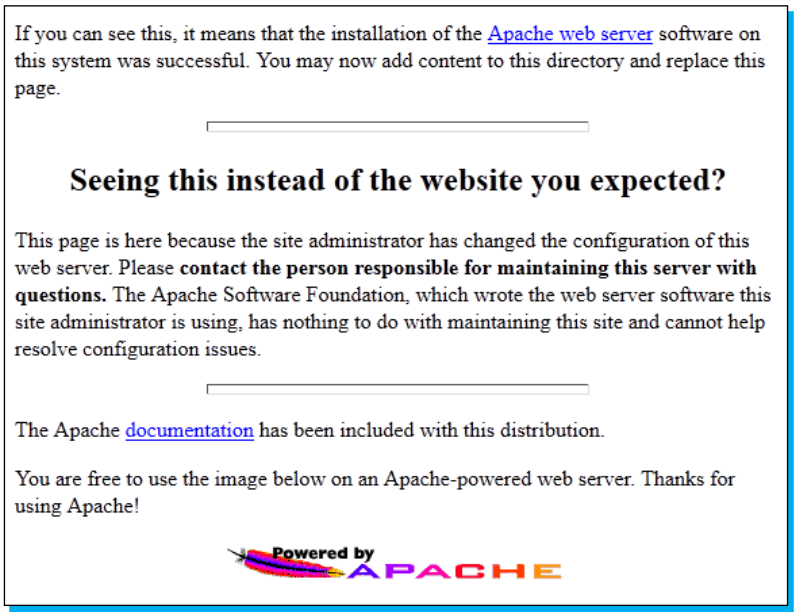
The URLs that provide information for installing the Apache web server on various operating systems are provided as follows:

- The instructions for installing Apache on Windows can be found at `http://httpd.apache.org/docs/current/platform/windows.html`

- The instructions for installing Apache on Linux can be found at `http://httpd.apache.org/docs/current/install.html`
- Mac OS X comes with a built-in Apache installation, which you can enable by navigating to **System Preferences | Sharing**, and selecting the **Web Sharing** checkbox

Install Apache or any other web server in your system. Browse to `http://yourwebserver.com` or `http://localhost`, and check that the installation is successful.

The `http://yourwebserver.com` link will show something similar to the the following screenshot, which confirms that Apache is installed successfully:



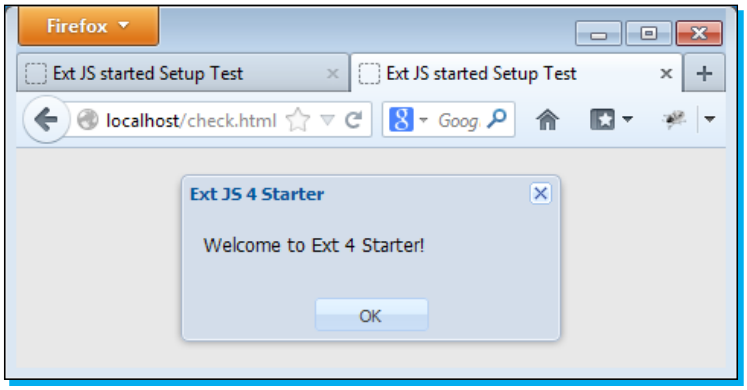
Step 4 – Unpacking Ext JS

In this tutorial, we will use Apache for Windows. Unpack the Ext JS bundle into the web server's root directory (`htdocs`). Rename the `Ext JS` folder with long version numbers to `extjs4` for simplicity. The root directory varies, depending upon your operating system and web server.

The Apache root directory path for various operating system are as follows:

- **Windows**: `C:\Program Files\Apache Software Foundation\Apache2.2\htdocs`
- **Linux**: `/var/www/`
- **Mac OS X**: `/Library/WebServer/Documents/`

The downloaded EXT JS bundle is packed with examples along with required sources. Browse to `http://yourwebserver.com/extjs4`, and make sure that it loads the Ext JS index page. This page provides access to all the examples to play around with the API. The **API Docs** link at bottom-right of the page lists the API information with a search text field at the top-right side of the page. As we progress through the tutorial, please refer to the API as and when required:



Step 5 –Testing Ext JS library.

A basic Ext JS application page will have a link tag with an Ext JS CSS file (`ext-all.css`), a script tag for the Ext JS library, and scripts related to your own application. In this example, we don't have any application-specific JavaScripts.

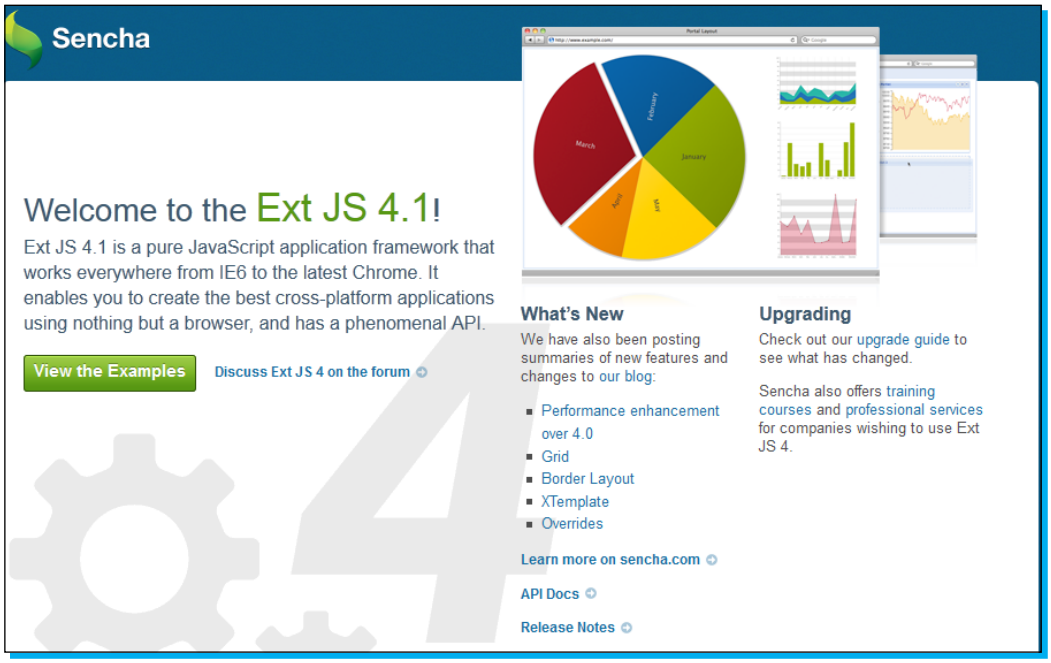
Create an HTML file named `check.html` with the code that follows beneath the `httpd` folder.

`Ext.onReady` is a method, which is executed when all the scripts are fully loaded. `Ext.Msg.alert` is a message box that shows a message to the user. The first parameter is the title and the second parameter is the message:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Ext JS started Setup Test</title>

<link rel="stylesheet" type="text/css" href="../extjs4/resources/css/
ext-all.css"></link>

<script type="text/javascript" src="../extjs4/ext-all-dev.js"></
script>

<script type="text/javascript">

Ext.onReady(function() {
    Ext.Msg.alert("Ext JS 4 Starter","Welcome to Ext 4 Starter!");
```

```
    });

</script>

</head>

<body>
</body>

</html>
```

The following screenshot shows `check.html` in action:



And that's it

By now, you should have a working installation of Ext JS, and should be able to play around and discover more about it.

Quick start – Creating and defining your Ext JS application

This section will explain how to create an application, define and create a class, and load the class on demand and a few other EXT JS built-in basic functions.

Step 1 – Preparing for an Ext JS application

The core library is packaged in three different ways:

- `ext.js`: This is the production version, which is minified and compressed
- `ext-debug.js`: This is the unminified development version
- `ext-dev.js`: This is the unminified development version, but outputs detailed error messages and warnings

To start with, create the following.

- A folder called `ExtJSStarter` under the `httpd` root folder
- Three folders named `app`, `data`, and `resources` respectively
- An `index.html` file, from where the application will be hosted
- A `starerApp.js` file through which all the application level JavaScript files are dynamically loaded on an as-needed basis

The resulting structure will be as follows:

httpd

- `extjs4` (unpacked ExtJS Framework and renamed to `extjs4`)
- `ExtJSStarter`
    - `app` (folder for application-specific JavaScript files)
    - `data` (folder for data)
    - `resources` (folder for images and CSS files)
    - `starterApp.js`
    - `index.html`

The core library is packaged in three different ways:

- `ext.js`: This is the production version, which is minified and compressed
- `ext-debug.js`: This is the unminified development version
- `ext-dev.js`: This is the unminified development version, but outputs detailed error messages and warnings

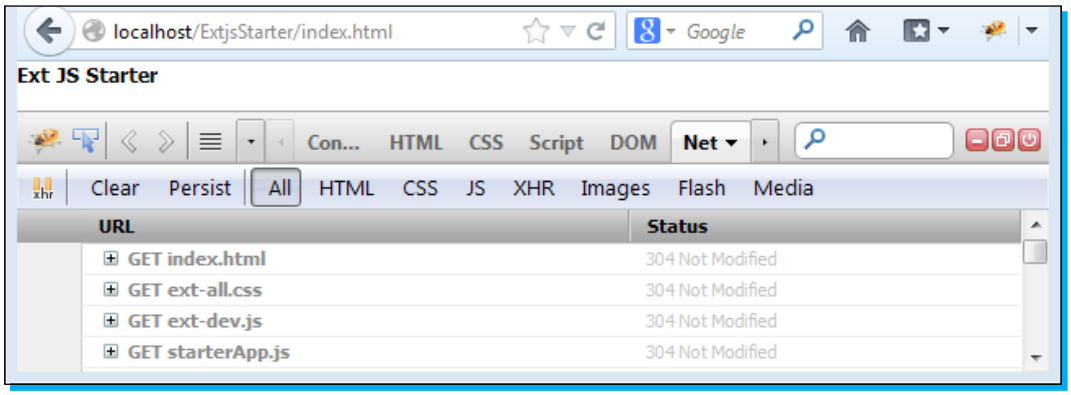
A basic Ext JS application page will have a `link` tag with an Ext JS CSS file (`ext-all.css`), a `script` tag for `ext-dev.js`, and scripts related to your own application:

```
<!DOCTYPE>
<html>
<head>
<title>Ext JS started Index Page </title>
<meta name="Author" content="Nagarajan Bhava">
<link rel="stylesheet" type="text/css"
href="../ext-4.1.1a/resources/css/ext-all.css"></link>
<script type="text/javascript" src="../../extjs4/ext-dev.js">
</script>
<script type="text/javascript" src="../ExtJSStarter/starterApp.js">
</script>
</head>
<body>
  <h2> Ext JS Starter <h2>
</body>
</html>
```

The Ext JS library can also be loaded alternatively from CDN:

- `http://cdn.sencha.io/ext-4.0.7-gpl/ext-all.js`
- `http://cdn.sencha.io/ext-4.0.7-gpl/resources/css/ext-all.css`

The following screenshot shows the index page as loaded in the browser with the Firebug having the **Net** tab open. The bottom Firebug debugger pane is made visible by clicking on the Firebug bug icon at the top-right corner of the browser window. It shows a list of files the browser has requested:



Step 2 – The MVC architecture

Model View Controller (**MVC**) is a well-known software architecture pattern, which separates the representation of information from the user's interaction with it. Ext JS comes with this architecture by default, and defines the architecture as follows.

- **Model**: It is collection of data fields, which persist through the data package. A model can be linked to another model using association and data steams, using proxy. It updates views when the data changes.
- **View**: A view requests the information from the model that it needs to generate an output representation.
- **Controller**: It holds application logic. It also instantiates models, stores, and views. It listens for events and modifies models and views.

Ext JS provides controllers with the `refs` property to gain reference to components inside the app, and a `control` method to take action based on the events that the referenced components fire. The *Top features you will want to know about* section deals in detail about handling events with controllers.

Step 3 – Defining your Ext JS application

Every Ext JS application starts with a call to `Ext.application()` to provide global settings, root namespace, references to other JavaScript files, and a `launch` function to execute the application.

Inside `starterApp.js`, define an application named `app` for the global namespace.

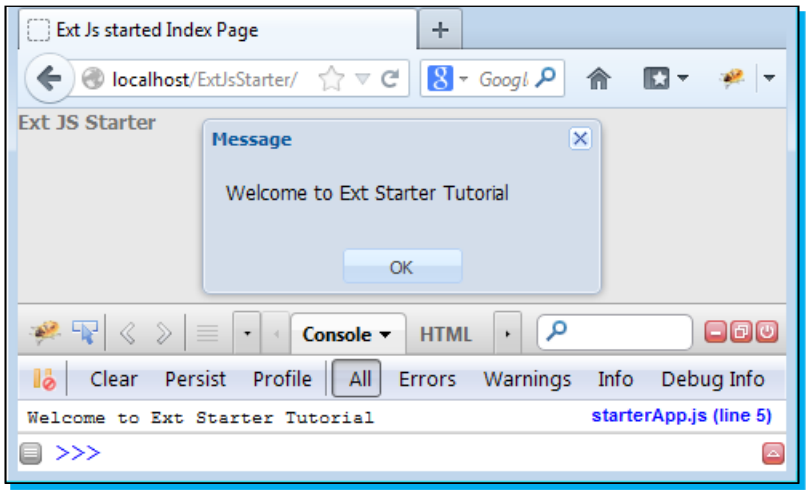
Define a `launch` function that outputs `Welcome to Ext Starter Tutorial` in the console and in the alert box.

The Firebug console appears similar to the following code snippet:

```
Ext.application({
  name :'app',
requires:['Ext.window.MessageBox'],
  launch:function(){
    console.log("Welcome to Ext Starter Tutorial");
    Ext.Msg.alert("Message"," Welcome to Ext Starter Tutorial");
  }
});
```

The preceding code snippet creates a global variable named `app`, and adds a `requires` property to load the Ext JS message box class on demand. All that the Ext JS application uses is only a single global variable. All application classes reside in this single namespace. When the page is ready, it calls the `launch` function and executes the `console.log` and `Ext.Msg.alert()` statement.

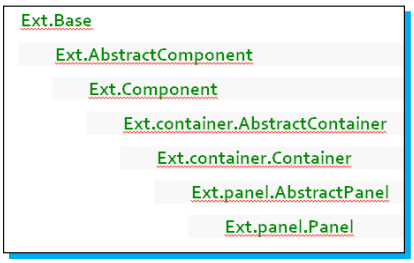
Using namespaces in JavaScript is a recommended practice to reduce the number of objects and functions that are added to the global scope in our applications. Namespaces are not built into the language, but the same benefits can be achieved by creating a single object (that is, `app`) and adding all our objects and functions to this object. It also helps to avoid the object collision, when using one JavaScript library with another one:



Step 4 – Understanding the Ext JS class hierarchy

The Ext JS framework is bundled with many well-organized classes that provide encapsulation and strong typing. Many classes in Ext JS are implemented as singletons with a set of static methods.

The class hierarchy starts from `Ext.Base` and drills down to various other class/components. A panel component is defined as `Ext.panel.panel`, and the class hierarchy will be similar to the following:

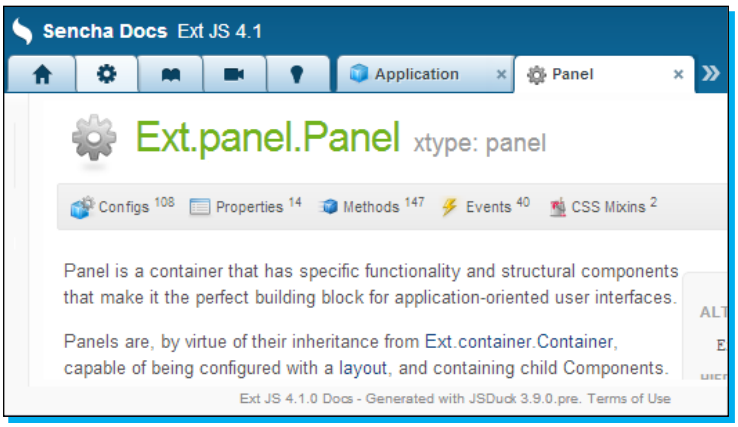


Please refer to the following links for API reference:

- **Offline API**: `http://yourservername.com/extjs4/ docs/index.html`
- **Online API**: `http://docs.sencha.com/ext-js/4-0/`

The API provides an easy "search" text field to search for a particular component. Searching for `panel` reveals a list of items that matches the word `panel`. Clicking on **Ext.panel.Panel** loads the panel API document. The top section shows links to configs, properties, methods, and events supported by the `panel` class. Explore all the provided options and get familiar with its usage.

As an example, search for the `Ext.` application that was used to launch an application in the previous step in the **Search** field, and get familiar with the API:



The names of the classes map directly to the file paths in which they are stored. The panel class is stored at `http://yourservername.com/extjs4/src/panel/Panel.js`. The link `http://yourservername.com/extjs4/src` is mapped to the `Ext` class.

Step 5 – Defining and creating the Ext JS class

A class in Ext JS 4 is defined using `Ext.define()`. `Ext.define()` detects and creates a new namespace as required, extends a class, and defers the creation of a new class if the class being extended has not been defined.

A class definition for `app.extJsBook` with its properties can be illustrated by the following code snippet:

```
Ext.define('app.extJsBook'){
  config:{
    title:'',
    price:'',
    author:''
  },
  constructor:function(config){
    this.initConfig(config)
  }
};
```

The preceding code snippet is explained as follows:

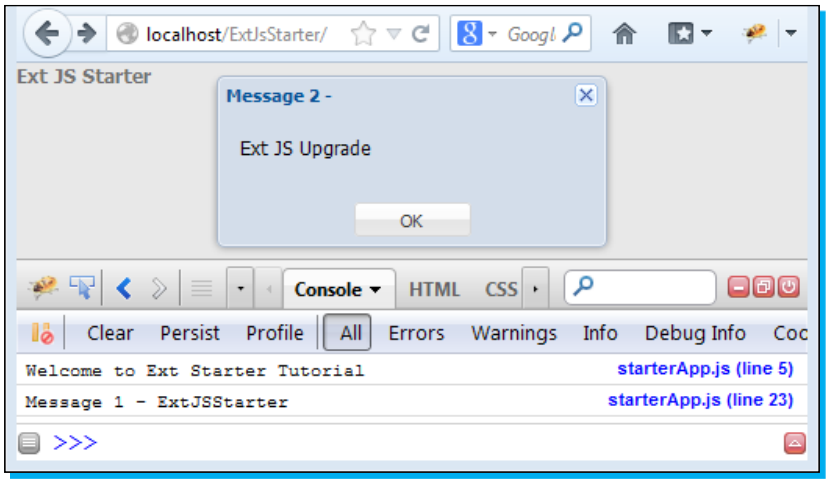
- `Config`: This allows to specify parameters that can be changed while instantiation
- `Constructor`: This is called during instantiation of the objects
- `initConfig(config)`: This creates `setter` and `getter` methods for config properties (that is, `setTitle()`, `getTitle()`, `setprice()`, `getPrice()`, `setAuthor()`, and `getAuthor()`)

The `app.extJsBook` class can be instantiated by using the `Ext.create()` method as illustrated in the following code snippet:

```
//Setting values using setter methods.

var book = Ext.create('app.extjsBook');
book.setTitle("ExtJSStarter");
book.setPrice("10$");
book.setAuthor('John');

console.log("Message 1 - ",book.getTitle());
//This message will be overridden by the second message
Ext.Msg.alert("Message 1 - ",book.getTitle());

// Setting values during instantiation

var book2 = Ext.create('app.extjsBook',{
  title:'Ext JS Upgrade',
  author:'Alex',
  price:'20$'
});
Ext.Msg.alert("Message 2 - ",book2.getTitle());
```

Add the code snippet of class definition and creation inside the `launch` method of `Ext.application` that was created in step 3.

Executing `index.html` will output something similar to the following screenshot. Please note that the message box shown is only the third message, and all previous messages have been overridden but are shown in the console for clarity:



To understand the component architecture, let's move the class definition to a separate file called `extjsBook` under the `ExtJSStarter/apps` directory. Add the name of the `app.extjsBook` class to the `requires` attribute of the application to load the class dynamically:

```
Ext.application({
  name:'app',
  requires:['Ext.window.MessageBox',
       'app.extjsBook'],
  launch:function(){
    console.log("Welcome to Ext Starter Tutorial");
    Ext.Msg.alert("Message","Welcome to Ext Starter Tutorial");

    var book = Ext.create('app.extjsBook');
    book.setTitle("ExtJSStarter");
    book.setPrice("10$");
    book.setAuthor('John');

    console.log("Message 1 - " +book.getTitle());
    Ext.Msg.alert("Message 2- ",book.getTitle());

    var book2 = Ext.create('app.extjsBook',{
      title:'Ext JS Upgrade',
      author:'Alex',
      price:'20$'
    });
    Ext.Msg.alert("Message 2 - ",book2.getTitle());
  }

});
```

`Ext.define()` provides two other important methods, which are `applyProperty()` and `updateProperty()`. The `apply` property is used to add additional logic to the setters, whereas the update `property` is used to do post processing or notification after the value has been set. In order to see both new and old values in the console, set `author` twice (that is, call `book.setAuthor()` in the application code twice).

The following code snippet changes the book title to uppercase and logs the new and old value of the author:

```
Ext.define('app.extjsBook',{
  config:{
    title:'',
    price:'',
    author:''
  },
  constructor:function(config){
    this.initConfig(config)
  },
// changing the title to upper case
  applyTitle:function(value){
    return Ext.String.capitalize(value.toUpperCase());
  },
//Logging both new and old value
  updateAuthor:function(newValue, oldValue){
    console.log("NewVal- " +newValue +":"+ "Old Val- " +oldValue);  }
});
```

The highlighted code to capitalize the author name uses `Ext.String.capitalize()`, an Ext JS version of the native JavaScript `String` utility class.

A class's functionality can also be added to another class using the `mixins` property. A static method can be added using the `static` property.

Step 6 – Instantiating Ext JS components

Components can be created either by using the `Ext.create()` method or by using the component's `xtype` property as a member of the parent component's `items` config.

- `Ext.container.AbstractContainer` in the class hierarchy provides the `items` configuration property to all its component sub-classes to add a single item, or an array of child components to the specified container.
- `Ext.Abstractcomponent` provides the `xtype` property, which is a shorter alternative to creating objects than using a full class name. Using `xtype` is the most common way to define component instances, especially in a container.

`Ext.create` is just like using the `new` keyword, and it will create the class right away, whereas with `xtype`, the class will not be created until it is needed. This is called **lazy-instantiation**. A hidden tab or panel with components laid using `xtype` will not be loaded until the tab or panel is visible, thereby increasing the performance of the page load time.

For example, the following code snippet illustrates a panel with 5 px padding on all four sides and 300 px width, containing text fields and two custom buttons rendered in the document's DOM body using the `renderTo` property:

```
var extjsBookPanel = Ext.create('Ext.panel.Panel', {
      bodyPadding: 5,
      width: 300,
      title: 'extjsBookPanel',
      items: [{
          xtype: 'textfield',
          fieldLabel: 'Publish date'
        },
        Ext.create('app.extjsButton', {
          text: 'Custom button instantiated'+
             'using Ext.create()',
        }),  {
          xtype: 'extjsButton',
          text:'Custom button instantiated using xtype'            }],
      renderTo: Ext.getBody()
    });
```

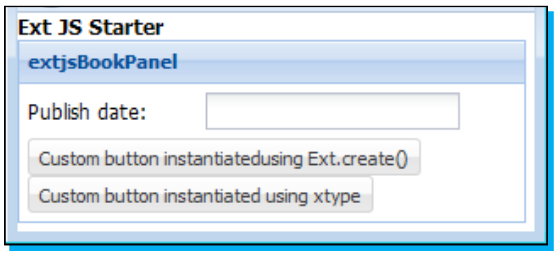
Three components have been included inside the item's configuration property. The `renderTo` property can be either an ID of an element or an element. `Ext.getBody()` returns the current document body as `Ext.Element`:

- A text field: A text field is instantiated using the `xtype` text field. Ext JS provides `xtype` for most of its available components.
- Custom button component 1: An `app.extjsButton` button component is instantiated using the `Ext.create()` method, as discussed in a previous step.
- Custom button component 2: An `app.extjsButton` button component is instantiated using a custom `xtype` name. In order for this to work, our `app.extjsBook` class definition needs to have `xtype` and `extend` properties specified along with a call to its parent `Button` class, as follows:

```
Ext.define('app.extjsBook',{
  xtype:'extjsButton',
  extend:'Ext.button.Button',
  constructor:function(config){
```

```
        //Call Ext.button.Button parent class
           this.callParent(arguments);
         }
      });
```

Adding the preceding code inside the application's `launch` function and with the modified `app.extjsButton` class property to support `xtype` will display something similar to the following screenshot:



`xtype` for all EXT JS components can be found at `http://docs.sencha.com/ext-js/4-1/#!/api/Ext.enums.Widget`.

Step 7 – Using Ext JS Lang functions

Native JavaScript offers a variety of utility classes such as `Array`, `Date`, `Number`, `Object`, and `String`. Ext JS provides its own version of utility classes for a smooth and consistent development experience across all browsers. It also provides extra functionality that is not available in native JavaScript. The available functions offer additional benefits. These can be found under `Ext.Array`, `Ext.Date`, `Ext.Number`, and so on. Making use of these functions wherever applicable will help to concentrate on core application logic, saving development time spent working on conversion utilities. A developer-specific class can also be added to the `Ext` namespace using `Ext.Define` with specific methods:

- `Ext.Array.contains(array,item)` can check whether or not the given array contains the specified item.
- `Ext.Date` supports a broad range of date formats that comes in handy while working with dates.

  For example, `Ext.Date.format(new Date(), 'Y-m-d'))` can be used to show the current date in YEAR-MM-DD format.

- `Ext.Function` provides functions to manage the ability to call functions with more features and control.

  For example, `Ext.Function.defer()` can be used to call a function after the number of milliseconds specified, optionally in a specific scope.

- `Ext.Number` provides six methods to handle numeric values.

  For example, `Ext.Number.randomInt(10,20)` returns a random integer between 10 and 20.

- `Ext.Object` provides about 10 methods to query and manipulate objects and JSON data structures.

  For example, `Ext.Object.fromQueryString('bookName=ExtJs&type=ebook')` outputs `{bookName:'ExtJs', type:'ebook'}`.

- `Ext.String` provides useful methods to manipulate strings.

  For example, `htmlEncode(value)` converts certain characters (`&`, `<`, `>`, `'`, and `"`) to their HTML character equivalents, for literal display in web pages.

Top features you'll want to know about

This section will deal with various Ext JS containers, components, forms, controllers, layouts, event mechanism, data package, templates, and so on.

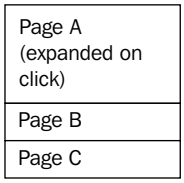
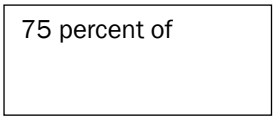
1 – Layout mechanism

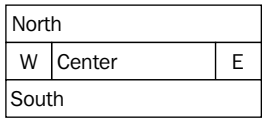
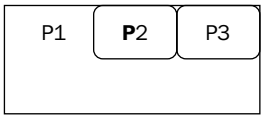
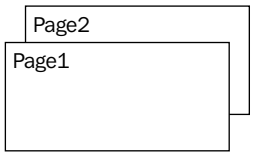
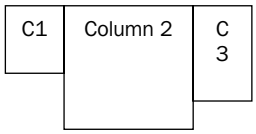
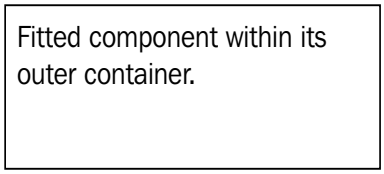
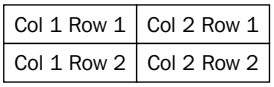
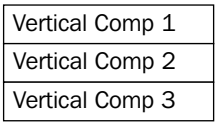
Ext JS container requires a layout to manage the sizing, positioning and layering of the child components. It provides different types of layouts to cater for most of the positioning requirements that anyone will come across.

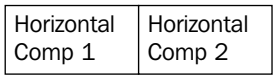
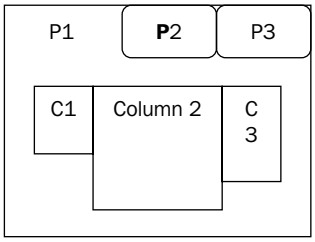
A simple HTML container with the `div` HTML element with CSS styling can be rendered easily with a few lines of code using the layout configuration property. The Ext JS layout mechanism takes care of generating the HTML element containers with clearly applied styles using the default CSS.

The following table lists the type of layouts, the component position depiction, and explains where it can be utilized. These layouts can be used either as they are, or combined with more than one, or in a nested fashion. Each layout provides specific configuration properties to size and position the child components. If the layout configuration is not specified, it uses a default layout manager that renders the components sequentially, one after the other. As we go through learning different components, different layouts will be shown to position the components.

The layouts and component depiction table is as follows:

Layout name	Component position depiction	Where to use?
Absolute		To position like CSS style positioning with X-Y coordinates
Accordion		To display one panel (that is, a page or a section) at a time in a stacked list of panels.
Anchor		To position the child components relative to their container.

Layout name	Component position depiction	Where to use?
Border		To lay different components on all the four sides (that is, north, south, east, and west) with the necessary center region. It provides collapse and resize behaviors.
Card (Tab Panel)		To create tabs for each section with tab names.
Card (Wizard)		To have a stack of pages one above the other, and show one page at a time hiding all others.
Column		To have content sections in different columns.
Fit		To stretch and fill the child component with its container.
Table		To create tables in the form of a standard HTML table.
vBox		To show the components vertically one below the other. It requires `height` for fixed height or `flex` to stretch the available height with respect to its container.

Layout name	Component position depiction	Where to use?
hBox		To show the components horizontally, one after another in a serial fashion. It requires `width` for a fixed width or `flex` to stretch to the available width with respect to its container.
Nested Layout		A column layout is nested inside a card (tab panel) layout.

The layouts in action can be visually seen at `http://dev.sencha.com/deploy/ext-4.0.0/examples/layout-browser/layout-browser.html`.

2 – Containers

Ext JS provides three types of containers:

- `Ext.container.Viewport (xtype:viewport)`: This is a special container that renders itself to the document body, thereby taking the whole browser viewable area. A page will only have one viewport and it does not provide scrolling.
- `Ext.container.Container (xtype: container)`: This is a top-level container class for most of the Ext JS components, such as panels and windows. It provides the addition, removal, and insertion behavior to all its children.
- `Ext.container.ButtonGroup (xtype:buttongroup)`: This container provides a way to arrange buttons in a group, in tabular format. This may be used when an application needs a task bar at the top or bottom of a panel.

Ext.container.Viewport

The viewport represents the viewable browser area equivalent to its available height and width. It renders itself to the page document body and resizes itself to the size of the browser. The viewport still requires layouts to position its child components. The following code snippet creates a viewport with a panel in it using `xtype` and the `fit` layout. Please note that the panel's `width` property has no meaning as the `fit` layout overrides the width and stretches the panel to its browser's window size. The code snippet shows a panel titled `Viewport Panel` and content `'A Panel inside viewport'`.

```
Ext.create("Ext.container.Viewport",{
layout:'fit',
items:[{
    xtype:'panel',
    width:300,
    html:'A Panel inside viewport',
    title:'Viewport Panel'
  }]
});
```

Ext.container.Container

A container can be created by just replacing the `xtype` property from `panel` to `container`. `container` is just a container without any title. The API for `container` does not have the `title` property defined as `container`, as it is just a container to lay and group child components and does not require any title.

The code snippet that follows, creates a container with a text field and a `submit` button with a margin of 5 px on its top, 5 px on its right, 0 px on its bottom, and 5 px on its left, and `vbox` (vertical box) layout.

Notice that a `style` property is added to the text field to provide a custom CSS specification margin bottom and the color. The `defaults` property is used to apply default settings to all its child items. In this case, `margin` is applied to both text fields and button.

Try replacing the `layout` property with `hbox` (horizontal box) instead of `vbox` and see the difference. The `hbox` layout lays the components horizontally one after the other:

```
Ext.create("Ext.container.Viewport",{
  layout:'fit',
  items:[{
    xtype:'container',
    // html:'A Panel inside viewport',
    defaults:{
      margin:'5 5 0 5'
    },
    layout:'vbox',
    items:[{
        xtype:'textfield',
        fieldLabel:'Name',
/*
* Custom CSS specification. The 'defaults' margin property need to be
* commented in order to see the effect of marginBottom.
*/
    style:{marginBottom:'20px',color:'blue'}
```

```
        },{
          xtype:'button',
          text:'submit'
        }]
      }]
});
```

The panel provides additional functions, such as including the store in the configuration, toolbar support on all four sides (`lbar`, `tbar`, `fbar`, and `rbar`), and it can be made floatable that cannot be found in container.

Ext.container.ButtonGroup

Button group provides a container for grouping the buttons and menus. Add the following code snippet to the `items` configuration of `Ext.container.Viewport` to visualize the button group. The following code snippet creates a button group type component with three buttons grouped together under the title `Days`. The `columns` property determines the number of column buttons:

```
{
  xtype:'buttongroup',
  columns:3,
  title:'Days',
  items:[{xtype:'button',text:1},
     {xtype:'button',text:2},
     {xtype:'button',text:2}]
}
```

3 – Components and data package

Ext JS includes various components. This sections deals with the tab panel and grid panel along with the data package.

Tabpanel

Tabpanel is a container that uses header-less panels and creates a modified button for each panel to look like a tab. It utilizes a card layout to lay child components.

The code snippet that follows creates a tab panel of size 400 x 400 pixels, with two panels representing two tabs named `Page1` and `Page2` in the document's body. The first item does not have `xtype` specified, but ExtJS considers the default panel `xtype`. The `renderTo` property can be either a DOM element or the ID of the element. The `activeTab` property is used to set the active tab using its item index position. Uncomment `layout:'fit'` (comment both width and height) and observe that the tab panel takes up the whole browser. Uncomment `layout:'absolute'`, `x:'10'`, and `y:'10'` (leave `layout:'fit'` commented), and observe that the tab panel is 10 px, absolutely positioned 10 px at x and y:

```
Ext.create('Ext.tab.Panel', {
width: 400,
height: 400,
activeTab:0,
//layout:'fit',
  //layout:'absolute',
  //x:10, y:10,
   renderTo: document.body,
   items: [{
   title:'Page 1',
    html:'Page1'
    }, {
    xtype:'panel',
   title:'Page 2',
    html:'Page2'
  }]
});
```

The tab panel or panel can have docked items on all the four sides to place the toolbars or tab bars. Add the following code to the tab panel:

```
dockedItems:[{
    xtype:'toolbar', dock:'bottom',
    items:[{xtype:'button', text:'Docked Button'}]
  }],
```

The preceding code snippet creates a toolbar with a button docked at the bottom position and sets to the `dockedItems` property.

`Ext.tab.Panel` internally uses card layout in its implementation. `Ext.container.Container` can be used with a card layout when there is no requirement for tab buttons.

Container with card layout

Try creating a container with two items in the container, with a card layout as shown in the following code snippet. Only one item (that is, first panel titled `Page 1`) will be displayed as the second panel (`Page 2`) is stacked under the first one. Uncomment the line `cardComp.getLayout().setActiveItem(1)` to bring the second item (`Page 2`) to the front and hide all the others. The `getLayout()` method provides the card layout used by the container, and `setActiveItem(ComponentIndex)` shows the item at the specified index hiding all the other items:

```
var cardComp =Ext.create('Ext.container.Container', {
  layout:'card',
  width: 400,
  height: 400,
  renderTo: document.body,
  items: [{
    title: 'Page 1',
    html:'Page1'
  }, {
    title: 'Page 2',
    html:'Page2'

  }]
});
//Show the second item visible by uncommenting this line.
//cardComp.getLayout().setActiveItem(1);
```

Data package

The new ExtJS 4 data package enables you to retrieve, decode, and use the data in the application. The core classes of data package are `Model`, `Proxy`, and `Store`.

`Model` represents the data or records in a store. It also includes field definitions, data validation rules, other models' associations, and proxy information for connecting to the server. It can have five types of validation rules, such as `presence`, `length`, `format`, `inclusion`, and `exclusion`, and a field may have more than one validation rule defined. It can also have custom methods defined.

Under the `app/model` folder, define a model by extending the `Ext.data.Model` class as indicated in the following example and save it as `Users.js`. The code snippet provides a custom method `capitalizeName` and adds the `length(to 2)` validation rule for the field `age`. It attaches the Ajax proxy to model and binds the model to a store:

```
Ext.define('app.model.Users',{
  extend:'Ext.data.Model',
  idProperty:'name', //default to id
  fields: [
```

```
        {name: 'name',type:'string'},
        {name:'email'},
        {name:'age',type:'int'},
        {name:'employed',type:'boolean'}
    ],
    capitalizeName:function(){
      this.set('name',Ext.util.Format.uppercase(this.get('name')));
    },
    validations:[
      {type:'length',field:'age',min:2}
    ],
});
```

Include the new component class to the application's `require` property in `starterApp.js`, whenever the Firebug console complains.

`Proxy` handles the loading and saving of data. It defines the connection between the browser and the server. It stores the load data via proxy into an array of data models. There are four types of server proxies, namely Ajax, REST, JSONP, and Direct, and three types of client proxies, namely Local storage, In-memory and Session storage. We will configure the Ajax `type` proxy in this tutorial:

- **Ajax**: It sends requests to a server on the same domain.
- **JSONP**: It sends requests to a different domain by dynamically instantiating the `script` tag.
- **REST**: It sends requests via a restful backend.
- **Direct**: It sends requests using `Ext.direct.Manager`. It provides a facility to invoke server methods directly from the ExtJS code as if it is calling directly from JavaScript. Please refer to `http://www.sencha.com/products/extjs/extdirect` for more info.
- **LocalStorage**: It uses the new HTML 5 localStorage API to save `Model` data locally on the client browser. HTML 5 local storage is a way for web pages to store named key/value pairs locally, within the client web browser.
- **In-memory**: It uses local variables to store data and hence the data is lost upon page refresh.
- **Session storage**: It uses HTML 5 Session storage for storing and retrieving data via a unique ID.

DirectJEngine (`http://code.google.com/p/directjngine/`) provides the infrastructure to invoke server e-Java classes directly from the Ext JS API.

Add the following configuration property to the preceding model:

```
proxy:{
  type:'ajax',
  url:'/ExtjsStarter/data/Users.json'
}
```

Under Apache's `htdocs/ExtJSStarter/data` folder, create a file called `Users.json` and add the following JSON data:

```
[{name:'David',email:'david@motorola.com',age:32,employed:true},
{name:'Jonny',email:'jonny@sencha.com',age:20,employed:false},
{name:'Anthony',email:'anthony@google.com',age:42.5,employed:true},
{name:'Steffi',email:'steffi@facebook.com',age:26,employed:true}]
```

The store class `Ext.data.Store` encapsulates the client-side data of the `Model` objects. It provides various methods to query, insert, remove, load, sort, and manage records. Add the following code below the model to bind the model to the new store and auto load upon creation. The loaded store contains the model bounded data from `Users.json`:

```
var extJS_UserStore = Ext.create('Ext.data.Store',{
  model:'app.model.Users',
  autoLoad:true
});
```

Let's make use of the created store in a grid panel to visualize the data in the browser in the section that follows.

Grid panel

Grids are used extensively to show a large amount of data in a tabular fashion, with sorting and filtering capabilities. ExtJS provided a robust grid component with various models for manipulating rows and columns. Grid comprises a store and a set of grid columns to render.

Create a grid panel with four columns, each associated with a `dataIndex` property. The column header `dataIndex` property uniquely identifies what records from the store should be added to each column. The `store` property defines a loaded store. The grid panel needs to be rendered to a DOM element in order to see it in the browser:

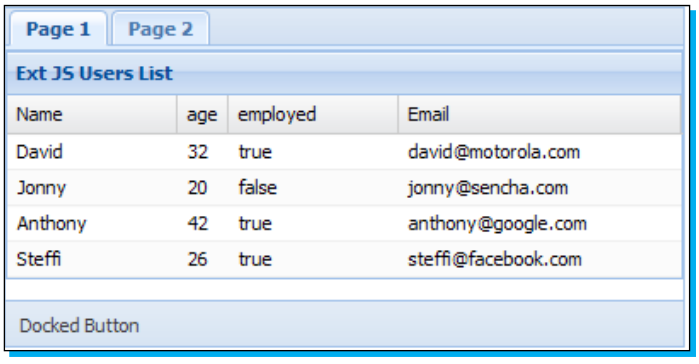
```
var extJS_UserGrid = Ext.create('Ext.grid.Panel',{
  title:'Ext JS Users List',
  store:extJS_UserStore, //created in previous chapter
  columns:[
    {header:'Name',dataIndex:'name'},
    {header:'Age',dataIndex:'age',width:30},
    {header:'Employed',dataIndex:'employed'},
    {header:'Email',dataIndex:'email',flex:1}
```

```
    ]

});
```

Let's add the grid to the first panel (that is, `Page 1`) of the tab panel created. Please notice that a grid panel is added to the first panel item (that is, `Page 1`):

```
Ext.create('Ext.tab.Panel', {
  width: 400,
 height: 200,
  dockedItems:[{
    xtype:'toolbar',
    dock:'bottom',
    items:[{text:'Docked Button'}]
  }],
  renderTo: document.body,
  items: [{
   title: 'Page 1',
    items:[
      extJS_UserGrid
    ]
  }, {
    xtype:'panel',
   title: 'Page 2',
    html:'Page2'

  }]
});
```

Loading the code in the browser will show something similar to the next screenshot. It shows two panels (**Page 1** and **Page 2**). The **Page 1** tab is active by default and it shows the grid panel with four columns and the loaded JSON data from the store using the AJAX proxy. Try clicking on the header and observe that the columns can be sorted:



Infinite scrolling

Infinite scrolling/paging of a grid to scroll through thousands of records without the performance penalties of rendering all the records on the screen at once can be done using the `verticalScroller` property.

Floating components: Any component subclass can be made floatable by using the `floating:true config` property. Floating components are managed by `ZindexManager` and are constrained to their parent containers.

4 – Traversing Ext JS components

Ex JS provides two ways to query the components, one way by using `Ext.ComponentManager` globally and the other way within `Ext.container.Container` on the document with a CSS-like syntax selector.

Components can be retrieved using `id`, `itemId`, and `xtype`, including the selector and operators for pattern matching.

Consider an outer panel with a text field, a button, and an inner panel with an inner text field in it, as follows:

```
Ext.create('Ext.Panel',{
  title:'ExtJS',
  id:'outerPanel',
  items:[
  {
    xtype:'textfield',
    id:'outerTextId',
  },{
    xtype:'button',
    text:'Submit',
    id:'buttonId'
  },{
    xtype:'panel',
    id:'innerPanel',
    items:[{
      xtype:'textfield',
      fieldLabel:'Country',
      id:'innerTextId',
      cls:'innerTextCls'
    }]
  }]
});
```

`Ext.ComponentQuery.query("#id')` returns an array, whereas `Ext.getCmp('id')` returns an object. `Ext.ComponentQuery.query("#id') [0]` and `Ext.getCmp('id')` are identical.

The following table illustrates the use of different ways to select and traverse the field using different methods available:

Selection methods	Return component object

`Ext.getCmp('outerPanel').down('button');`
`Ext.getCmp('outerTextId') .nextSibling()`
`Ext.getCmp('outerTextId').next()`
`Ext.getCmp('outerTextId').nextNode();`
	Button

`Ext.getCmp('buttonId').previousSibling()` and `Ext.getCmp('buttonId').prev()`	Outer text field
`Ext.ComponentQuery.query('textfield[cls=innerTextCls]')`	Inner text field
`Ext.getCmp('outerTextId').up()`	Outer panel – form
`Ext.getCmp('innerPanel').down()`	Inner text field
`Ext.getCmp('innerTextId').` `findParentByType('panel')`	Inner panel
`Ext.getCmp('buttonId').` `findParentByType('panel')`	Outer panel

5 – Working with forms

A form in Ext JS refers to an `Ext.form.Panel` class that provides a standard container for all the `form` elements. It automatically creates a `BasicForm` object for managing any `Ext.form.field.Field` objects that are added to the container. By default, the form panel is configured with an anchor layout for its child items, and can be changed if required. It provides various configuration properties and methods to manage and handle the fields.

Ext JS 4 supports the following form fields:

- Checkbox and checkbox group
- Combo box
- Display field (display only text field) and hidden
- HTML editor
- Date and number
- Radio and radio group

- Slider and spinner (fields with a pair of up/down spinner arrow buttons)
- Text, trigger (text wrapper that adds clickable buttons), and text area
- Time and file

All fields provide getter and setter methods, which are events and methods for tracking and validating values. A few important basic `form` methods that are used frequently are provided as follows:

- `getFieldValues`: It returns all the field values in the form of name-value pairs
- `getFields`: It returns all the fields
- `getValues`: It returns all the string field values in the form name-value pairs
- `findField('id')`: It finds a specific Ext JS fields using its ID
- `isValid`: It checks for validity
- `markInvalid(error)`: It marks the field as invalid

The `form` fields can be enabled and disabled on validation change using the `formBind` and `disabled` properties. The preceding code snippet illustrates a form panel with a text field that is validated against a non-empty string length between 2 and 10, a checkbox, and submit button that gets enabled only when the text field value is valid.

The validation rules on text fields are defined by specifying the following properties:

- `allowBlank`, `maskRe`, `regex`, and `stripCharsRe`
- `maxLength` and `minLength`
- `validator`, `validationChange`, and `vType`

Validation on a field happens by default and can be deferred by setting the `validateOnChange` property to `false` and invoking validation by using the `validate()`, `isValid()`, and `hasInvalidField()` methods.

The validation error messages are customized using `blanktext`, `maxlengthText`, `regexText`, and `vTypeText`.

`vType`: `Ext.form.field.vTypes` is a virtual type implementation that defines ExtJS-specific default and custom validation (that is, custom `vType` for user specific validation) rules. Ext JS provides default `vType` properties, such as `email`, `URL`, `alpha`, and `alphanum`.

The following code snippet illustrates a form panel with a text field, a checkbox, and a button for AJAX submission. The highlighted lines provide the validation rules for the fields via properties. The `Ext.form.Basic.submit()` button posts the data to an application server using the HTTP `post` action to the URL specified in `url` property and alerts the user with a success or failure message.

The code snippet for the form panel is as follows:

```
{
  xtype:'form',bodyPadding:10,
  title:'A Form Panel',
  width:300,height:400,
  url:'http://someurl.com/extjs.jsp',
  items:[{
    xtype:'textfield',
    fieldLabel:'Name',
    vType:'alpha', //only alphabets are allowed
    allowBlank:false, minLength:2, maxLength:10
  } ,{
    xtype:'checkbox',
    boxLabel:'I agree terms and conditions',
    name:'termsCB',
    formBind:true,disabled:true
  }],
  buttons:[{
    text:'Submit',
    formBind:true,
    disabled:true,
    handler:function(){
      var form = this.up('form').getForm();
      if(form.isValid()){
        form.submit({
          success:function(form,action){
          Ext.Msg.alert("Success".action.result.msg");
          },
          failure:function(form,action){
            Ext.Msg.alert("Success".action.result.msg);
          }
        });  //submit end
      }//form isValid end

    }}]//buttons end
  }]
}
```

The preceding form panel code is processed by the following server response:

```
//_________ Server Response _____________________________
/**
Note: Please be informed that the response can be of JSON type also.
Hence failure case can be categorized with ids and can be reported
accordingly.
{
  "success":true,
  "msg":"Data Submitted Successfully"
}
{
  "success":false,
  "msg":"Data Submission failed"
}
**/
```

A Basic Form's `getValue()` method returns only the string values and `getFieldValue()` returns all type-specific data values (for example, `Date` objects for date fields, `Boolean` for combo boxes, and so on)

6 – Templates

`Ext.XTemplate` is used to do the following:

- Generating HTML output from arrays and store data
- Basic conditional processing (`for` and `if` statements) and custom member support
- Executing inline code with built-in template variables

A simple template can be defined using the `Ext.XTemplate` constructor by passing an HTML fragment (with fields in curly braces as an argument, as illustrated in the following code snippet). It also provides a `<tpl>` tag for operators to repeat the `template` block for an array and an `if` operator for conditional processing. It can also have member functions.

The following code snippet creates a template and applies the object and array to it:

```
var tplObj = Ext.create('Ext.XTemplate',
  '<div> Hi {user}, How are you?</br> Are you from {city}</div>');
tplObj.compile();
tplObj.applyTemplate({user:'Tony',city:'London'}) //Object

var tplArray = Ext.create('Ext.XTemplate',
    '<div> Hi {0}, How are you?</br> You are from {1}</div>');
tplArray.compile();
```

```
tplArray.applyTemplate(['Tony','London']) //Array

<tpl for="xxx">
  //HTML fragment
</tpl>

</tpl>
```

The `html` fragment can have either the fields or the array index inside curly braces, as shown in the preceding code snippet. The `applyTemplate` method replaces the `{user}` and the `{city}` values in the template. The template values can be an array (if the parameters are numeric) or an object.

Check out more on template examples at `http://docs.sencha.com/ext-js/4-1/#!/api/Ext.XTemplate`.

7 – Controllers and events

Controllers are responsible for events from various components that occur within our app. Let's create a grid view controller (using the grid panel created in the *Quick start – Creating and defining your Ext JS application* section), a new app, and an HTML file that loads the app and checks how the events are handled in the controller. It provides a central location to manage all the events of the application components.

Create a file called `app/view/ extjsUsersListView.js`, and add the following code to it. Please note that the store defined in the grid is already loaded using `app.model.Users` (`app/model/users`) referred in *Data package* section:

```
Ext.define('app.view.extjsUsersListView',{
  extend: 'Ext.grid.Panel',
  alias : 'widget.extjsUsersList',
  title : 'Ext JS Users',
  initComponent: function() {
    this.store=extJS_UserStore,
    this.columns=[
      {header:'Name',dataIndex:'name'},
      {header:'age',dataIndex:'age',width:30},
      {header:'employed',dataIndex:'employed'},
      {header:'Email',dataIndex:'email',flex:1}
    ]
   this.callParent(arguments);
  }
});
```

Create a file called `app/controller/ extjsUsersController.js`, and add the following code to it:

The following code snippet defines a controller `app.controller. extjsUsersController` by extending `Ext.app.Controller`. The controller's `views` property is configured with the view created in previous section. The `init()` method defines how the events are handled in the controller through the `control` function. This function can either refer to the references or the xtypes. The reference `'viewport > panel'` uses the `Ext.ComponentQuery` syntax to point to the UI components and finds the direct child panel inside the viewport. The `'extjsUsersList'` xtype refers to the grid panel (that is, `view extjsUsersListView'`) for which we want to control the `itemclick` and `afterrender` events:

```
Ext.define('app.controller. extjsUsersController, {
 extend: 'Ext.app.Controller',
  views: [
    extjsUsersListView'
  ],
  init: function() {
   console.log('Initialized!');

    this.control({
      // Events of panel referred through Ext.ComponentQuery
     'viewport > panel': {
      //This event is fired, as the grid panel extends panel.
        render: this.onPanelRendered
      },
      // Events of grid panel referred through its xtype
      'extjsUsersList': {
      itemclick: this.editUser,
        afterrender:this.gridAfterRender
     }
    });
  },
  onPanelRendered: function() {
    console.log('The panel was rendered');
  },
  gridAfterRender: function(grid, opts) {
   console.log('Grid Component rendered');
  },
  editUser: function(grid, record) {
   console.log('Clicked on ' + record.get('name'));
  }
})
```

'alias' is a property and refers to a short name of a custom class for defining xtypes using widget.<xtypeName>. For example, widget.usergrid and widget.userPanel.

Create an application file mvcApp.js under the ExtJSStarter folder and add the following code to it. The controllers property in the application is configured with extjsUsersController, which is created in the previous section, and the grid component (extjsUsersList) is added to the viewport. Please refer to the alias property of the grid component for the xtype name mentioned:

```
Ext.application({
  name:'app',
  requires:['Ext.window.MessageBox','app.model.Users',
    'app.controller.extjsUsersController',
    'app.view.extjsUsersListView',
    'Ext.container.Viewport','Ext.grid.Panel'
  ],
  controllers: [
    'extjsUsersController'
  ],
  launch:function(){
     Ext.create('Ext.container.Viewport', {
     items: [
        {xtype:'extjsUsersList' }
      ]
    });
  }
});
```

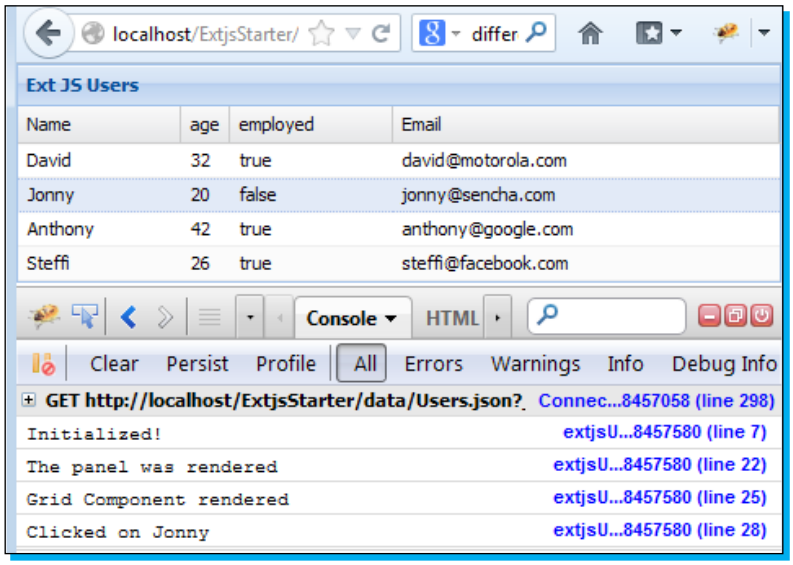
Create a file called mvcApp.html under the ExtJSStarter folder, and add the following code to it:

```
<html>
<head>
<title>Ext Js started Index Page </title>
<meta name="Author" content="Nagarajan Bhava">
<link rel="stylesheet" type="text/css"     href="../extjs4/resources/
css/ext-all.css"></link>
<script type="text/javascript" src="../../extjs4/ext-    dev.js"></
script>
<script type="text/javascript"     src="../ExtJSStarter/data/Users.
json"></script>
 <script type="text/javascript"     src="../ExtJSStarter/mvcApp.js"></
script>
</head>
```

```
<body>
</body>
</html>
```

Loading `http://yourservername.com/ExtJSStarter/mvcApp.html` with the Firebug debugger on will show something similar to the next screenshot.

The initialized application renders the panel and grid panel, and listens to the `itemclick` event. When the page is loaded, the application's `init` method is called and followed by the rendering of panel. Clicking any rows will fire the `itemclick` event and log the message in the console:



8 – Event handling

Events in Ext JS are either DOM or JavaScript events. Event sources in Ext JS are an extension of the `Ext.Observable` class. The event properties are as follows:

- **Event** is a message sent (fired) by an event source to inform listeners that something happened in the view
- **Event source** is an object that can fire events
- **Event listener** is a function that is called when an event source fires an event
- Events can be fired using the `on` function

DOM events: Browsers that display HTML pages watch for user actions and fire events if the actions are occurring on DOM elements. Ext JS's `Ext.Element` function wraps DOM elements together with their events, and provides a way to call the same event handlers in a different way:

```
//Basic way
<div id="divId" onclick="alert('You clicked me')">Click Here!</div>
//Ext JS way
 Ext.get(divId).on('click', function() {alert('You clicked me');});
```

JavaScript Events: During development, it is normal to have events from JavaScript objects other than DOM events. The following code snippet add a listener to Ext JS panel.

```
//Create panel
var myPanel = new Ext.Panel({...});
// Add event "some_Event"
myPanel.on('some_event', function(…) {…});
```

Custom events can be added by using the `addEvents()` and `fireEvent()` methods, as shown in the following code snippet:

```
MyPanel = Ext.extend(Ext.Panel, {
  initComponent:function() {
    ...
     this.callParent(arguments);
    ...
    // add custom events
    this.addEvents('loaded');
  },
  load:function(cfg) {
    ....
    // fire loaded event
    this.fireEvent('loaded', this, cfg);
  }
});
```

Events can be relayed from the outcome of specified observations of the preceding code if they are fired by its execution. Assuming that the content panel mentioned in the preceding section is inside a window (`win`), it will tell the window to handle the `'loaded'` event from the content panel:

```
win.relayEvents(MyPanel, ['loaded']);
win.on( {
   'loaded' : function (panel, obj) {...},
   scope : this
});
```

9 – Other advanced features

A few other advanced features are listed as follows:

- **JSBuilder**: This is a cross-platform Java application that allows you to customize your JavaScript and CSS files for building your projects. It uses JSON-based (`http://en.wikipedia.org/wiki/JSON`) configurations and the YUI compressor (`http://yui.github.com/yuicompressor/`) for minification.
- **Drawing and charting**: Ext JS provides a robust HTML 5 based drawing and charting packages that enables one to create cross-browser and cross-device graphics in a versatile way.
- **Theming**: Ext JS 4 has a brand-new theming system to customize the look of your application while still supporting all browsers.
- **HTML5 local storage**: The local storage proxy `Ext.data.proxy.LocalStorage` uses the new HTML5 `localStorage` API to save `Model` data locally on the client browser. HTML5 local storage is a way for web pages to store named key/value pairs locally, within the client web browser. Like cookies, this data persists even after you navigate away from the website, close your browser tab, or exit your browser.
- **Animation**: CSS 3 based animations using keyframes can be made using the `Ext.fx.Animator` class.
- **History management**: History management using `Ext.util.History` allows you to register arbitrary application states on navigation actions, which can later be used to reset the application when the user navigates forward or backward through the browser history.

People and places you should get to know

This tutorial has been created from the experience gained using references from Ext JS APIs, forums, tutorials, and guides from Sencha's official site `http://www.sencha.com/`.

If you need help with *Instant Ext JS Starter*, here are some people and places that will prove invaluable.

Official sites

The official sites for download, training, and FAQs are as follows:

- **Ext JS**: `http://www.sencha.com`
- **Training**: `http://www.sencha.com/training/`
- **Download**: `http://www.sencha.com/products/extjs/download/`
- **Licencing FAQs**: `http://www.sencha.com/legal/open-source-faq/`

Articles and tutorials

The following list provides useful articles and tutorials to help you to explore more about the framework:

- **API**: `http://docs.sencha.com/ext-js/4-0/`
- **Examples**: `http://docs.sencha.com/ext-js/4-0/#!/example`
- **Getting started guide**: `http://docs.sencha.com/ext-js/4-0/#/guide/getting_started`
- **Saki's (An example page from an active Ext JS forum member) page**: `http://examples.extjs.eu/`
- **Theming**: `http://docs.sencha.com/ext-js/4-0/#!/guide/theming` and `http://docs.sencha.com/ext-js/4-0/#!/video/19159630`
- **Drawing and charting**: `http://docs.sencha.com/ext-js/4-0/#!/guide/drawing_and_charting`

Community

If you are in need of any help, then forums and support will come handy:

- **Sencha official forum**: The registration can be done at `http://www.sencha.com/forum/register.php`
- **Ext JS 4 Forum**: `http://www.sencha.com/forum/forumdisplay.php?79-Ext-JS-Community-Forums-4.x`
- **Sencha Support**: `http://www.sencha.com/support/`
- **User FAQs**: `http://www.sencha.com/support/faq/`

Links

A few links for building JavaScripts, CSS, and unit testing are as follows:

- **Chrome developer tools page**: `https://developers.google.com/chrome-developer-tools/docs/overview`
- **HTML5 offline storage**: `http://docs.sencha.com/ext-js/4-0/#!/video/17844271`
- **Unit testing with Jasmine**: `http://docs.sencha.com/ext-js/4-0/#!/guide/testing`
- **JSBuilder (Javascript and CSS builder tool)**: `http://www.sencha.com/products/jsbuilder`

Books

The following books provide an insight into the internals of JavaScript and help with deep understanding of the core of EXT JS or any other JavaScript framework:

- *JavaScript: The Good Parts*, *Douglas Crockford*, *O'Reilly Media/Yahoo Press*
- *JavaScript Patterns*, *Stoyan Stefanov*, *O'Reilly Media*
- *Pro JavaScript Techniques*, *John Resig*, *Apress*

Blogs

Jozef Sakalos, also known as Saki, is one of the leading forum members, who actively participates in many queries and provides excellent solutions. You can find his blog at `http://blog.extjs.eu/`.

Twitter

Follow Aaron Conron, who leads the Sencha Architect team on `https://twitter.com/aconran`.

Thank you for buying
Instant Ext JS Starter

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] PUBLISHING

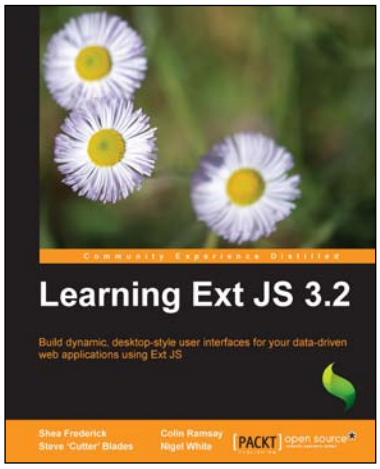
Ext JS 4 Web Application Development Cookbook

ISBN: 978-1-84951-686-0 Paperback: 488 pages

Over 110 easy-to-follow recipes backed up with real-life examples, walking you through the basic Ext JS features to advanced application design using Sencha Ext JS

1. Learn how to build Rich Internet Applications with the latest version of the Ext JS framework in a cookbook style
2. From creating forms to theming your interface, you will learn the building blocks for developing the perfect web application
3. Easy to follow recipes step through practical and detailed examples which are all fully backed up with code, illustrations, and tips

---



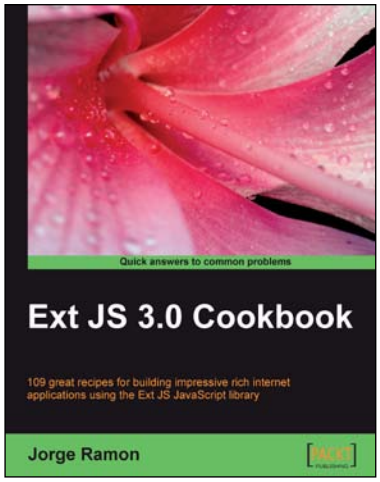
Learning Ext JS 3.2

ISBN: 978-1-84951-120-9 Paperback: 432 pages

Build dynamic, desktop-style user interfaces for your data-driven web applications using Ext JS

1. Learn to build consistent, attractive web interfaces with the framework components
2. Integrate your existing data and web services with Ext JS data support
3. Enhance your JavaScript skills by using Ext's DOM and AJAX helpers
4. Extend Ext JS through custom components

[PACKT] PUBLISHING



Ext JS 3.0 Cookbook

ISBN: 978-1-84719-870-9 Paperback: 376 pages

Clear step-by-step recipes for building impressive rich internet applications using the Ext JS JavaScript library

1. Master the Ext JS widgets and learn to create custom components to suit your needs
2. Build striking native and custom layouts, forms, grids, list views, tree views, charts, tab panels, menus, toolbars and much more for your real-world user interfaces
3. Packed with easy-to-follow examples to exercise all of the features of the Ext JS library

---

Mastering Ext JS: RAW

ISBN: 978-1-78216-400-5 Paperback: 350 pages

Discover how you can take your skills with Ext JS to the next level

1. Learn expert tips and tricks to make your web applications look stunning
2. Full of engaging practical examples specifically tailored to augment your skills
3. Build a series of great themes, login pages, and menus