FREE eBook

LEARNING extjs

Free unaffiliated eBook created from
Stack Overflow contributors.

#extjs

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: extjs

It is an unofficial and free extjs ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official extjs.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with extjs

Remarks

ExtJS is a JavaScript framework from Sencha for building Rich Internet Applications. It boasts one of the largest libraries of pre-built modular UI components.

Since version 5.0, Sencha has advocated the use of Model-View-ViewModel (MVVM) architecture on its platform. It also maintains support for Model-View-Controller (MVC) architecture which was the primary architecture style supported up through version 4.x.

Additionally, Sencha has focused on outfitting ExtJS with mobile-centric and responsive web application capabilities. Its former Sencha Touch framework has been integrated with ExtJS since version 6.0 with efforts to combine the customer bases and consolidate redundancies in the new combined framework.

Versions

Version	Release Date
6.2	2016-06-14
— 6.0	2015-08-28
5.0	2014-06-02
4.0	2011-04-26
3.0	2009-07-06
2.0	2007-12-04
1.1	2007-04-15

Examples

Creating a Hello World Application – Via Sencha Cmd

This example demonstrates creating a basic application in ExtJS using Sencha Cmd to bootstrap the process - this method will automatically generate some code and a skeleton structure for the project.

Open a console window and change the working directory to an appropriate space in which to work. In the same window and directory run the following command to generate a new application.

```
> sencha -sdk /path/to/ext-sdk generate app HelloWorld  ./HelloWorld
```

Note: The `-sdk` flag specifies the location of the directory extracted from the framework archive.

In ExtJS 6+ Sencha have merged both the ExtJS and Touch frameworks into a single codebase, differentiated by the terms ***classic*** and ***modern*** respectively. For simplicity if you do not wish to target mobile devices, an additional flag may be specified in the command to reduce clutter in the workspace.

```
> sencha -sdk /path/to/ext-sdk generate app -classic HelloWorld ./HelloWorld
```

Without any further configuration, a fully functional demo application should now reside in the local directory. Now change the working directory to the new `HelloWorld` project directory and run:

```
> sencha app watch
```

By doing this, the project is compiled using the default build profile and a simple HTTP server is started which allows the viewing of the application locally through a web browser. By default on port **1841**.

Installation & Setup

Typical usage of ExtJS leverages the framework to build single-page rich-applications (RIA). The simplest way to get started is to make use of Sencha Cmd, a CLI build tool covering most of the general concerns in a deployment life-cycle, primarily:

- package and dependency management
- code compilation / bundling and minification
- managing build strategies for different targets and platforms

» Download Sencha Cmd

The second step is download the SDK, ExtJS is a commercial product - to obtain a copy, one of:

- login to Sencha Support for the commercially licences version (product page)
- apply for an evaluation copy which will be valid for 30 days
- request the GPL v3 version available for open-source projects
  (note that you may not be able to access the latest version with this option)

After downloading the SDK ensure the archive is extracted before proceeding.

Note: See the official Getting Started documentation for a comprehensive guide to ExtJS projects.

After installing Sencha Cmd, it's availability can be verified by opening a console window and running:

```
> sencha help
```

We now have the tools necessary to create and deploy ExtJS applications, take note of the directory location where the SDK was extracted as this will be required in further examples.

Creating a Hello World Application – Manually

Let's start using ExtJS to build a simple web application.

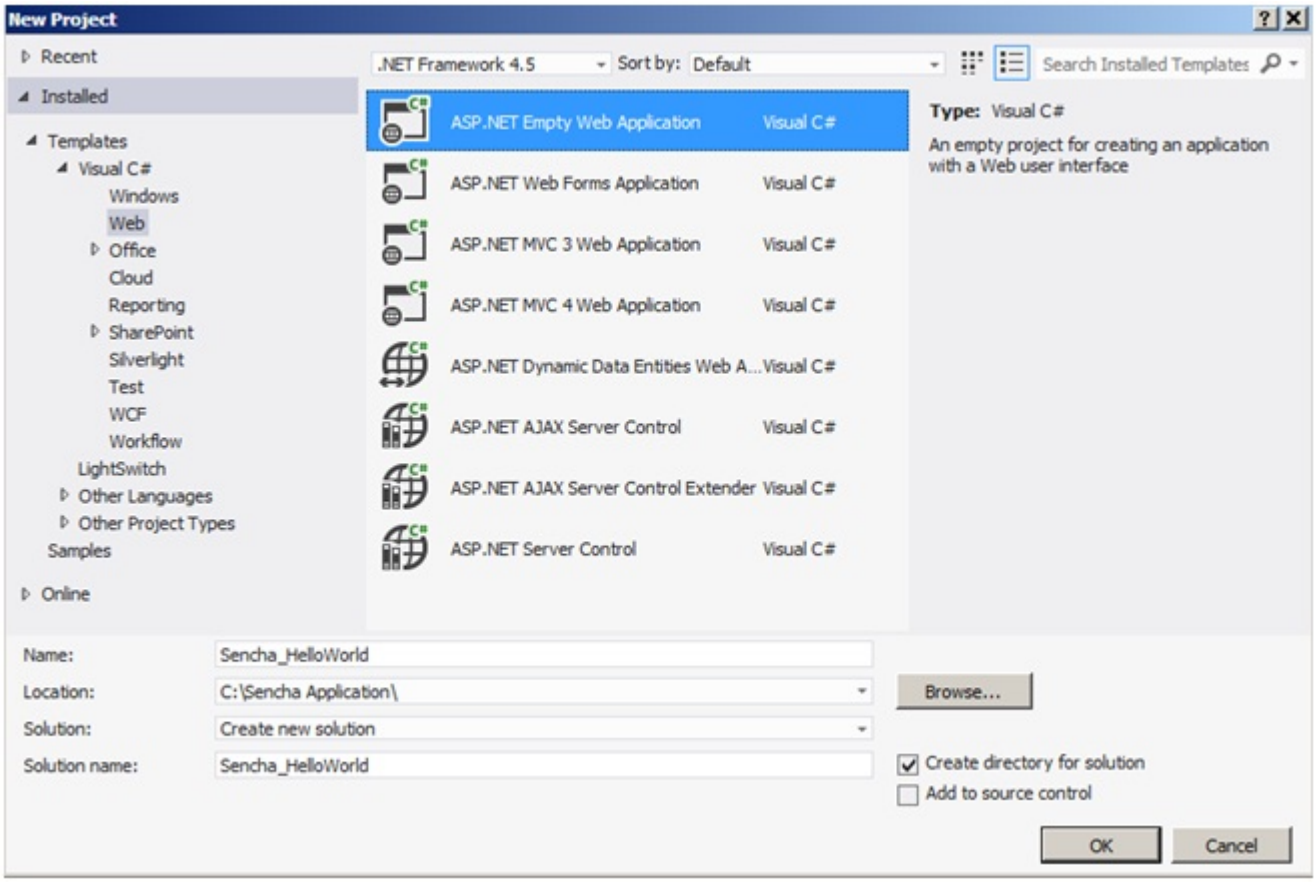
We will create a simple web application which will have only one physical page (aspx/html). At a minimum, every ExtJS application will contain one HTML and one JavaScript file—usually index.html and app.js.

The file index.html or your default page will include the references to the CSS and JavaScript code of ExtJS, along with your app.js file containing the code for your application (basically starting point of your web application).

Let’s create a simple web application that will use ExtJS library components:

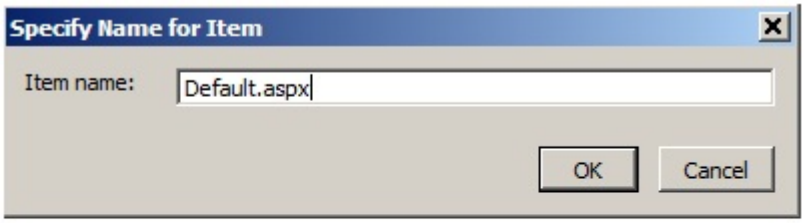
Step 1: Create a empty web application

As shown in the screenshot, I have created an empty web application. To make it simple, you can use any web application project in the editor or IDE of your choice.



Step 2: Add a default web page

If you have created an empty web application, then we need to include a web page that would be the starting page of our application.



Step 3: Add Ext Js References to Default.aspx

This step shows how we make use of extJS Library. As shown in the screenshot in the Default.aspx, I have just referred 3 files:

- ext-all.js
- ext-all.css
- app.js

Sencha has partnered with CacheFly, a global content network, to provide free CDN hosting for the ExtJS framework. In this sample I have used Ext's CDN library, however we could use the same files (ext-all.js & ext-all.css) from our project directory instead or as backups in the event the CDN was unavailable.

By referring to the app.js, it would be loaded into the browser and it would be the starting point for our application.

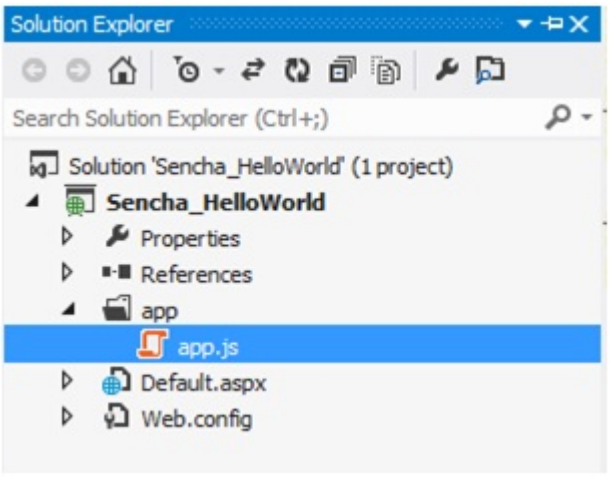
Apart from these files, we have a placeholder where UI will be rendered. In this sample, we have a div with id “whitespace” that we will use later to render UI.


```
<script type="text/javascript" src="http://cdn.sencha.com/ext/trial/5.0.0/build/ext-
all.js"></script>

<link rel="stylesheet" type="text/css"
```

```
href="http://cdn.sencha.com/ext/trial/5.0.0/build/packages/ext-theme-
neptune/build/resources/ext-theme-neptune-all.css"/>

<script src="app/app.js"></script>
```

Step 4: Add app folder & app.js in your web project

ExtJS provides us with a way to manage the code in an MVC pattern. As shown in the screenshot, we have a container folder for our ExtJS application, in this case 'app'. This folder will contain all of our application code split into various folders, i.e., model, view, controller, store, etc. Currently, it has only the app.js file.



Step 5: Write your code in app.js

App.js is the starting point of our application; for this sample I have just used minimum configuration required to launch the application.

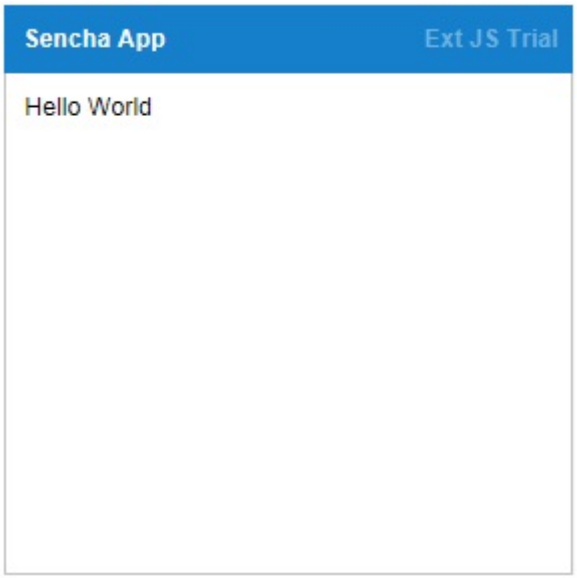
Ext.application represents an ExtJS application which does several things. It creates a global variable '**SenchaApp**' provided in the name configuration and all of the application classes (models, views, controllers, stores) will reside in the single namespace. Launch is a function that is called automatically when all the application is ready (all the classes are loaded properly).

In this sample, we are creating a Panel with some configuration and rendering it on the placeholder that we provided in the *Default.aspx.*

```
Ext.application({
    name: 'SenchaApp',
    launch: function () {
        Ext.create('Ext.panel.Panel', {
            title: 'Sencha App',
            width: 300,
            height: 300,
            bodyPadding:10,
            renderTo: 'whitespace',
            html:'Hello World'
        });
    }
});
```

Output Screenshot

When you run this web application with Default.aspx as a startup page, the following window will appear in the browser.



Read Getting started with extjs online: https://riptutorial.com/extjs/topic/819/getting-started-with-extjs

Chapter 2: Common Pitfalls & Best Practices

Examples

Separation of Concerns

Worse

ViewController:

```
// ...
myMethod: function () {
    this.getView().lookup('myhappyfield').setValue(100);
}
//...
```

View:

```
//...
items: [
    {
        xtype: 'textfield',
        reference: 'myhappyfield'
    }
]
//...
```

Better

ViewController:

```
// ...
myMethod: function () {
    this.getView().setHappiness(100);
}
//...
```

View:

```
//...
items: [
    {
        xtype: 'textfield',
        reference: 'myhappyfield'
    }
],
setHappiness: function (happiness) {
    this.lookup('myhappyfield').setValue(happiness);
}
//...
```

Explanation

In this example, the two snippets of code perform the same task. However, in the event the reference to `myhappyfield` changes or the methodology of indicating 'happiness' changes significantly, the former approach requires changes each place the reference is used.

With separated concerns (the latter example), the view provides an abstracted way to modify 'happiness' that other classes may use. The querying and component manipulation are kept in one place (right alongside the view definition itself!) and the calls to the abstracted method need not change.

Although it is possible for a controller to allow querying down through the layers of a view, it is strongly advisable to abstract that behavior into methods on the view. In this way, a view can provide standardized ways for other classes to influence it and minimize or eliminate changes to other classes when the structure of a view changes.

Extend vs Override

Overrides:

Override file:

```
Ext.define('MyApp.override.CornField',
    override: 'Ext.form.field.Text',
    initComponent: function () {
        this.callParent(arguments);
        this.setValue('Corn!');
    }
);
```

Use in app:

```
{
    xtype: 'textfield'
}
```

Extensions:

Override file:

```
Ext.define('MyApp.form.field.CornField',
    extend: 'Ext.form.field.Text',
    alias: 'widget.cornfield',
    initComponent: function () {
        this.callParent(arguments);
        this.setValue('Corn!');
    }
);
```

Use in app:

```
{
    xtype: 'cornfield'
}
```

Explanation

ExtJS provides two main ways to change the behavior of existing classes: extending them, and overriding them. Each has benefits and pitfalls that should be considered before using them.

Extensions

Extending a class creates a new class that inherits its behavior and configuration from its parent. By creating a new class through extension, repeated configuration and behavioral changes can be made in a central location and reused throughout the application. The biggest advantage of extension is that the parent class remains intact and available for simpler use cases where the extended behavior is not desired.

Examples of good use cases for extensions include custom form fields with special behavior, specialized modals, and custom components in general.

Overrides

Overriding a class modifies the behavior of an existing class in place. Configuration and methods in overrides replace their parent class counterparts entirely, creating new default configurations and behavior that populate throughout the application. Overrides should be used sparingly because of the destructive nature of their use; an extended class can typically provide the same benefits while leaving the parent class undisturbed.

However, overrides can provide benefits in some situations. Good use cases include fixing bugs in existing classes, modifying proxy behavior to append extra information to requests, such as a token or session data, and generally forcing a specific behavior to be the default behavior across an application.

Separate Overrides from Bug Fixes

In ExtJS, you can override nearly any method of the framework and replace it with your own. This allows you to modify existing classes without directly modifying the ExtJS source code.

Sometimes, you may want to enhance an existing class or provide a sane default property on a class.

For example, you may want all of your model data fields to allow null values.

```
Ext.define('MyApp.override.DataField', {
  override: 'Ext.data.field.Field',
  allowNull: true
});
```

In other cases, you will need to fix something that is broken in the framework.

Here is an example of a bug fix with documentation. Note that the classname contains "fix" rather than "override". The actual name isn't important, but the separation is.

```
Ext.define('MyApp.fix.FieldBase', {
  override: 'Ext.form.field.Base',
  /**
   * Add a description of what this fix does.
   * Be sure to add URLs to important reference information!
   *
   * You can also include some of your own tags to help identify
   * when the problem started and what Sencha bug ticket it relates to.
   *
   * @extversion 5.1.1
   * @extbug EXTJS-15302
   */
  publishValue: function () {
    this.publishState('value', this.getValue());
  }
});
```

Now when it comes time to upgrade to the next version of ExtJS, there is only one place you need to check to see which of your bug fixes can be removed.

Read Common Pitfalls & Best Practices online: https://riptutorial.com/extjs/topic/5412/common-pitfalls---best-practices

Chapter 3: Event Model

Introduction

ExtJS advocates the use of firing of and listening for events between classes. By firing events and listening for fired events, classes require no 'dirty' knowledge of each others' class structure and prevent coupling code together. Additionally, events make it easy to listen to multiple instances of the same component by allowing a generic listener for all objects with the same selector. Finally, other classes may also be able to make use of events that already exist.

Examples

Controllers Listening to Components

```
Ext.define('App.Duck', {
    extend: 'Ext.Component',
    alias: 'widget.duck',
    initComponent: function () {
        this.callParent(arguments);
        this._quack();
    },
    _quack: function () {
        console.log('The duck says "Quack!"');
        this.fireEvent('quack');
    },
    feed: function () {
        console.log('The duck looks content.');
    },
    poke: function () {
        this._quack();
    }
});

var feedController = Ext.create('Ext.app.Controller', {
    listen: {
        components: {
            duck: {
                quack: 'feedDuck'
            }
        }
    },
    feedDuck: function (duck) {
        duck.feed();
    }
});

var countController = Ext.create('Ext.app.Controller', {
    listen: {
        components: {
            duck: {
                quack: 'addCount'
            }
        }
```

```
    },
    quackCount: 0,
    addCount: function (duck) {
        this.quackCount++;
        console.log('There have been this many quacks: ' + this.quackCount);
    }
});

var firstDuck = Ext.create('App.Duck');
// The duck says "Quack!"
// The duck looks content.
// There have been this many quacks: 1
var secondDuck = Ext.create('App.Duck');
// The duck says "Quack!"
// The duck looks content.
// There have been this many quacks: 2
firstDuck.poke();
// The duck says "Quack!"
// The duck looks content.
// There have been this many quacks: 3
```

Read Event Model online: https://riptutorial.com/extjs/topic/9314/event-model

Chapter 4: ExtJS AJAX

Introduction

A singleton instance of an [`Ext.data.Connection`][1] class. This class is used to communicate with your server side.
[1]:http://docs.sencha.com/extjs/6.0.1/classic/src/Connection.js.html#Ext.data.Connection

Examples

Basic Request

Some of the Class properties `Ext.Data.Connection`

Properties	Details
`url`	Address of the request
`timeout`	Waiting time in milliseconds
`success`	Return on success
`failure`	Return on failure

```
Ext.Ajax.on("beforerequest", function(conn , options , eOpts) {
    console.log("beforerequest");
});
Ext.Ajax.on("requestcomplete", function(conn , response , options , eOpts) {
    console.log("requestcomplete");
});
Ext.Ajax.on("requestexception", function(conn , response , options , eOpts) {
    console.log("requestexception");
});

Ext.Ajax.request({
    url: 'mypath/sample.json',
    timeout: 60000,
    success: function(response, opts) {
        var obj = Ext.decode(response.responseText);
        console.log(obj);
    },
    failure: function(response, opts) {
        console.log('server-side failure with status code ' + response.status);
    }
});
```

Read ExtJS AJAX online: https://riptutorial.com/extjs/topic/8134/extjs-ajax

Chapter 5: MVC / MVVM - Application Architecture

Examples

Introduction to models

A model represents some data object in an application. For example you can have a model such as: Fruit, Car, Building, etc. in your application. Models are normally used by stores. Here is example how you would define a new model class. e.g.

```
Ext.define('MyApp.model.Person', {
    extend: 'Ext.data.Model',
    fields: [
        {name: 'name', type: 'string'},
        {name: 'surname', type: 'string'},
        {name: 'age', type: 'int'}
    ],

    getFullName: function() {
        return this.get('name') + " " + this.get('surname');
    }
});
```

After defining our model class we would possibly like to create an instance of it and probably call some methods. For example:

```
// Create person instance
var person = Ext.create('MyApp.model.Person', {
    name : 'Jon',
    surname: 'Doe',
    age  : 24
});

alert(person.getFullName()); // Display person full name
```

ExtJS 4 MVC CRUD App Example

Online demo is here: http://ext4all.com/post/extjs-4-mvc-application-architecture.html

Define a model:

```
// /scripts/app/model/User.js
Ext.define('AM.model.User', {
    extend: 'Ext.data.Model',
    fields: ['id', 'name', 'email']
});
```

Define a store with proxy:

```
// /scripts/app/store/Users.js
Ext.define('AM.store.Users', {
    extend: 'Ext.data.Store',
    model: 'AM.model.User',
    autoLoad: true,
    autoSync: true,
    proxy: {
        type: 'ajax',
        limitParam: 'size',
        startParam: undefined,
        api: {
            create: '/user/add',
            read: '/user/list',
            update: '/user/update',
            destroy: '/user/delete'
        },
        reader: {
            type: 'json',
            root: 'data',
            successProperty: 'success'
        },
        writer: {
            type: 'json',
            writeAllFields: false
        }
    }
});
```

Define add user view - it's a window with a form inside:

```
// /scripts/app/view/user/Add.js
Ext.define('AM.view.user.Add', {
    extend: 'Ext.window.Window',
    alias: 'widget.useradd',
    title: 'Add User',
    layout: 'fit',
    autoShow: true,
    initComponent: function () {
        this.items = [
            {
                xtype: 'form',
                bodyStyle: {
                    background: 'none',
                    padding: '10px',
                    border: '0'
                },
                items: [
                    {
                        xtype: 'textfield',
                        name: 'name',
                        allowBlank: false,
                        fieldLabel: 'Name'
                    },
                    {
                        xtype: 'textfield',
                        name: 'email',
                        allowBlank: false,
                        vtype: 'email',
                        fieldLabel: 'Email'
                    }
```

```
                ]
            }
        ];
        this.buttons = [
            {
                text: 'Save',
                action: 'save'
            },
            {
                text: 'Cancel',
                scope: this,
                handler: this.close
            }
        ];
        this.callParent(arguments);
    }
});
```

Define edit user view - it's also window with form inside:

```
// /scripts/app/view/user/Edit.js
Ext.define('AM.view.user.Edit', {
    extend: 'Ext.window.Window',
    alias: 'widget.useredit',
    title: 'Edit User',
    layout: 'fit',
    autoShow: true,
    initComponent: function () {
        this.items = [
            {
                xtype: 'form',
                bodyStyle: {
                    background: 'none',
                    padding: '10px',
                    border: '0'
                },
                items: [
                    {
                        xtype: 'textfield',
                        name: 'name',
                        allowBlank: false,
                        fieldLabel: 'Name'
                    },
                    {
                        xtype: 'textfield',
                        name: 'email',
                        allowBlank: false,
                        vtype: 'email',
                        fieldLabel: 'Email'
                    }
                ]
            }
        ];
        this.buttons = [
            {
                text: 'Save',
                action: 'save'
            },
            {
                text: 'Cancel',
```

```
                scope: this,
                handler: this.close
            }
        ];
        this.callParent(arguments);
    }
});
```

Define a user list view - it's a grid with columns Id, Name, Email

```
// /scripts/app/view/user/List.js
Ext.define('AM.view.user.List', {
    extend: 'Ext.grid.Panel',
    alias: 'widget.userlist',
    title: 'All Users',
    store: 'Users',
    initComponent: function () {
        this.tbar = [{
            text: 'Create User', action: 'create'
        }];
        this.columns = [
            { header: 'Id', dataIndex: 'id', width: 50 },
            { header: 'Name', dataIndex: 'name', flex: 1 },
            { header: 'Email', dataIndex: 'email', flex: 1 }
        ];
        this.addEvents('removeitem');
        this.actions = {
            removeitem: Ext.create('Ext.Action', {
                text: 'Remove User',
                handler: function () { this.fireEvent('removeitem', this.getSelected()) },
                scope: this
            })
        };
        var contextMenu = Ext.create('Ext.menu.Menu', {
            items: [
                this.actions.removeitem
            ]
        });
        this.on({
            itemcontextmenu: function (view, rec, node, index, e) {
                e.stopEvent();
                contextMenu.showAt(e.getXY());
                return false;
            }
        });
        this.callParent(arguments);
    },
    getSelected: function () {
        var sm = this.getSelectionModel();
        var rs = sm.getSelection();
        if (rs.length) {
            return rs[0];
        }
        return null;
    }
});
```

Define a controller to handle views events:

```
// /scripts/app/controller/Users.js
Ext.define('AM.controller.Users', {
    extend: 'Ext.app.Controller',
    stores: [
        'Users'
    ],
    models: [
        'User'
    ],
    views: [
        'user.List',
        'user.Add',
        'user.Edit'
    ],
    init: function () {
        this.control({
            'userlist': {
                itemdblclick: this.editUser,
                removeitem: this.removeUser
            },
            'userlist > toolbar > button[action=create]': {
                click: this.onCreateUser
            },
            'useradd button[action=save]': {
                click: this.doCreateUser
            },
            'useredit button[action=save]': {
                click: this.updateUser
            }
        });
    },
    editUser: function (grid, record) {
        var view = Ext.widget('useredit');
        view.down('form').loadRecord(record);
    },
    removeUser: function (user) {
        Ext.Msg.confirm('Remove User', 'Are you sure?', function (button) {
            if (button == 'yes') {
                this.getUsersStore().remove(user);
            }
        }, this);
    },
    onCreateUser: function () {
        var view = Ext.widget('useradd');
    },
    doCreateUser: function (button) {
        var win = button.up('window'),
            form = win.down('form'),
            values = form.getValues(),
            store = this.getUsersStore();
        if (form.getForm().isValid()) {
            store.add(values);
            win.close();
        }
    },
    updateUser: function (button) {
        var win = button.up('window'),
            form = win.down('form'),
            record = form.getRecord(),
            values = form.getValues(),
            store = this.getUsersStore();
```

```
        if (form.getForm().isValid()) {
            record.set(values);
            win.close();
        }
    }
});
```

Define your app in app.js:

```
// /scripts/app/app.js
Ext.Loader.setConfig({  enabled: true });

Ext.application({
    name: 'AM',
    appFolder: 'scripts/app',
    controllers: [
        'Users'
    ],
    launch: function () {
        Ext.create('Ext.container.Viewport', {
            layout: 'border',
            items: {
                xtype: 'userlist',
                region: 'center',
                margins: '5 5 5 5'
            }
        });
    }
});
```

Online demo is here: http://ext4all.com/post/extjs-4-mvc-application-architecture.html

Read MVC / MVVM - Application Architecture online: https://riptutorial.com/extjs/topic/3854/mvc---mvvm---application-architecture

Credits

S. No	Chapters	Contributors
1	Getting started with extjs	Ben Rhys-Lewis, Community, David Millar, Dharmesh Hadiyal, Emissary, srinivasarao, UDID
2	Common Pitfalls & Best Practices	David Millar, Emissary, John Krull
3	Event Model	David Millar
4	ExtJS AJAX	Alexandre N.
5	MVC / MVVM - Application Architecture	CD.., Emissary, Giorgi Moniava, khmurach