SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – II-Rich Internet Applications – SCS1401

II. Introduction

Functions - Objects and Classes - Javascript in Web Browsers - Getting started with Ext JS - Creating your first Ext JS Application - MVC Basics.

Introduction

Ext JS stands for Extended JavaScript. It is a JavaScript framework and a product of Sencha, based on YUI (Yahoo User Interface). It is basically a desktop application development platform with modern UI. Ext JS is a popular JavaScript framework which provides rich UI for building web applications with cross-browser functionality. Ext JS is basically used for creating desktop applications. It supports all the modern browsers such as IE6+, FF, Chrome, Safari 6+, Opera 12+, etc. Whereas another product of Sencha, Sencha Touch is used for mobile applications.

Ext JS is based on MVC/MVVM architecture. The latest version of Ext JS 6 is a single platform, which can be used for both desktop and mobile application without having different code for different platform.

MVC − Model View Controller architecture (version 4)

MVVM − Model View Viewmodel (version 5)

History

Ext JS 1.1

- The first version of Ext JS was developed by Jack Slocum in 2006.
- It was a set of utility classes, which is an extension of YUI.
- He named the library as YUI-ext.

Ext JS 2.0

- Ext JS version 2.0 was released in 2007.
- This version doesn’t have backward compatibility with previous version of Ext JS.

Ext JS 3.0

- Ext JS version 3.0 was released in 2009.
- This version added new features as chart and list view.
- It had backward compatibility with version 2.0.

Ext JS 4.0

- Ext JS version 4.0 was released in 2011.
- It had the complete revised structure, which was followed by MVC architecture and a speedy application.

Ext JS 5.0

- Ext JS version 5.0 was released in 2014.
- The major change in this release was to change the MVC architecture to MVVM architecture ability to build desktop apps on touch-enabled devices, two-way data binding, responsive layouts, and many more features.

Ext JS 6.0

- Released in July 1,2015.
- Ext JS 6 merges the Ext JS (for desktop application) and Sencha Touch (for mobile application) framework.

Ext JS 7.0

- Released in August 29,2019.
- The Ext JS 7.0 Modern toolkit supports: drag and drop, row editor and collapsible group for grids, accordion layout, form group, breadcrumb toolbar and drag and drop for tree view.
- Ext JS 7.0 has been upgraded to support Font Awesome 5+, which is SVG based.

Features of Ext Js

- Customizable UI widgets with collection of rich UI such as grids, pivot grids, forms, charts, trees.
- Code compatibility of new versions with the older one.
- A flexible layout manager helps to organize the display of data and content across multiple browsers, devices, and screen sizes.
- Advance data package decouples the UI widgets from the data layer.
- Sophisticated data visualization
- Rich Data Analytics

- Pre integrated and tested UI components
- Layout Manager and responsive configs.
- Ext JS is a client-side JavaScript application framework to build enterprise applications.
- Ext JS supports object-oriented programming concepts using JavaScript which makes easy application development and maintenance.
- Ext JS supports Single Page Application development.
- Ext JS includes MVC and MVVM architecture.
- Ext JS Data package makes it easy to retrieve or saving of data.
- Ext JS includes OOB UI components, containers and layouts.
- Ext JS includes drag and drop functionality for UI containers and components.
- Ext JS includes localization package that makes it easy to localize application.
- Includes OOB themes.

Advantages

- Streamlines cross-platform development across desktops, tablets, and smartphones — for both modern and legacy browsers.
- Increases the productivity of development teams by integrating into enterprise development environments via IDE plugins.
- Reduces the cost of web application development.
- It has set of widgets for making UI powerful and easy.
- It follows MVC architecture so highly readable code.

Limitations

- The size of the library is large, around 500 KB, which makes initial loading time more and makes application slow.
- HTML is full of tags that makes it complex and difficult to debug.
- It is free for open source applications but paid for commercial applications.
- Need quite experienced developer for developing Ext JS applications.

Major Browsers support by Ext Js

- IE 6+
- Mozilla firefox version 1.5+

- Apple safari version 2+
- Opera version 9+
- Chrome 10+

Downloading Library Files

- Download the trial version of Ext JS library files from Sencha https://www.sencha.com
- Unzip the folder and you will find various JavaScript and CSS files, which you will include in our application.
- JavaScript Files – JS file which you can find under the folder \ext-6.0.1-trial\ext6.0.1\build are

JavaScript Files

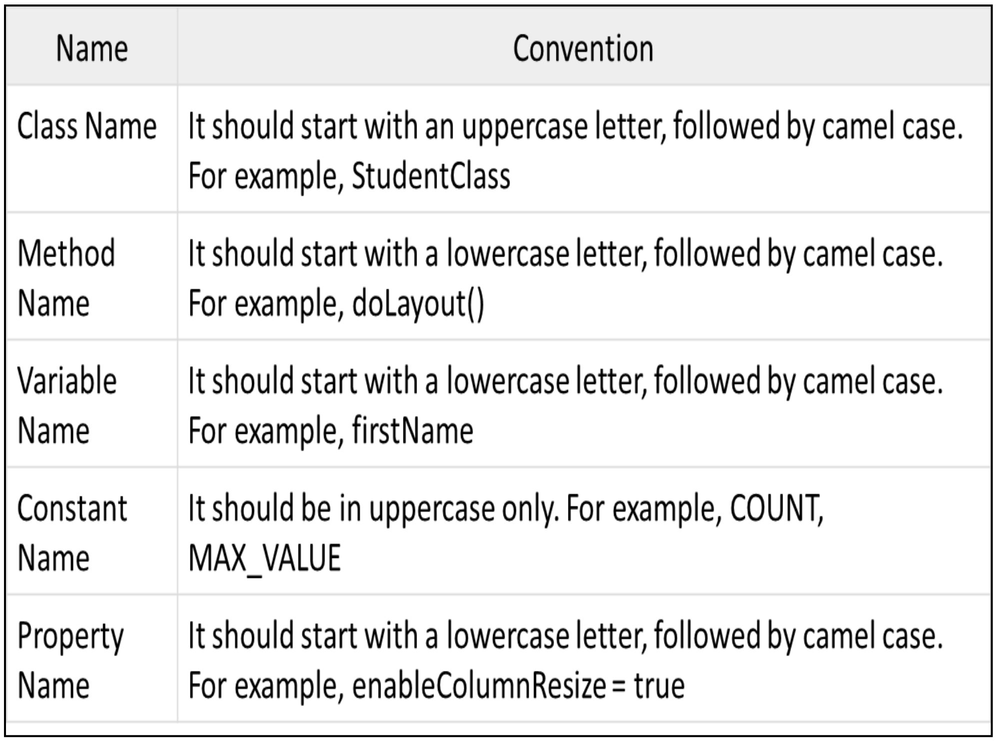
- **ext.js** -This is the core file which contains all the functionalities to run the application.
- **ext-all.js-**This file contains all the code minified with no comments in the file.
- **ext-all-debug.js-**This is the unminified version of ext-all.js for debugging purpose.
- **ext-all-dev.js-**This file is also unminified and is used for development purpose as it contains all the comments and console logs to check any errors/issue.
- **ext-all.js-**This file is used for production purpose mostly as it is much smaller than any other.

Difference between Ext.all and Ext.js

- **ext-all.js:** This file contains the entire Ext JS framework (used for Development & testing) **ext.js:** This file contains the minimum Ext JS code (Ext JS base library)- used in Production.

Naming Convention

- Naming convention is a set of rules to be followed for identifiers is shown in Table 2.1.
- It makes the code more readable and understandable to other programmers as well.
- The second word will start with an uppercase letter always.



Name	Convention
Class Name	It should start with an uppercase letter, followed by camel case. For example, StudentClass
Method Name	It should start with a lowercase letter, followed by camel case. For example, doLayout()
Variable Name	It should start with a lowercase letter, followed by camel case. For example, firstName
Constant Name	It should be in uppercase only. For example, COUNT, MAX_VALUE
Property Name	It should start with a lowercase letter, followed by camel case. For example, enableColumnResize = true

Table 2.1. Naming Convention

Architecture

----------src

----------resources

-------------------CSS files

-------------------Images

----------JavaScript

--------------------App Folder

-------------------------------Controller

------------------------------------Contoller.js

-------------------------------Model

------------------------------------Model.js

-------------------------------Store

------------------------------------Store.js

```
-------------------------------View
------------------------------------View.js
-------------------------------Utils
------------------------------------Utils.js
--------------------------------app.js
-----------HTML files
```

Ext Js App

- Ext JS app folder will reside in JavaScript folder of your project.
- The App will contain controller, view, model, store, and utility files with app.js.
- The App will contain **controller, view, model, store, and utility files** with app.js.

– **app.js** − The main file from where the flow of program will start, which should be included in the main HTML file using <script> tag.

Controller.js

– It is the controller file of Ext JS MVC architecture.

– This contains all the control of the application, the events listeners, and most of the functionality of the code.

– It has the path defined for all the other files used in that application such as store, view, model, require, mixins.

View.js

– It contains the interface part of the application, which shows up to the user.

– Ext JS uses various UI rich views, which can be extended and customized here according to the requirement.

Model.js

– It contains the objects which binds the store data to view.

– It has the mapping of backend data objects to the view dataIndex.

– The data is fetched with the help of store.

Store.js

- It contains the data locally cached which is to be rendered on the view with the help of model objects.
- Store fetches the data using proxies which has the path defined for services to fetch the backend data.

Utils.js

- It is not included in MVC architecture but a best practice to use to make the code clean, less complex, and more readable.
- In MVVM architecture, the controller is replaced by ViewModel.

Components of Ext JS

- The components can be defined as **the number of widgets, which helps to create an application.**
- The Component is referred to as the DOM element, which contains the complicated functionality.
- All components are inherited from the **Ext. Component** class.
- Ext JS application includes various types of elements, such as Combobox, grid, panel, container, textfield, numberfield, etc is shown in Figure.2.1.

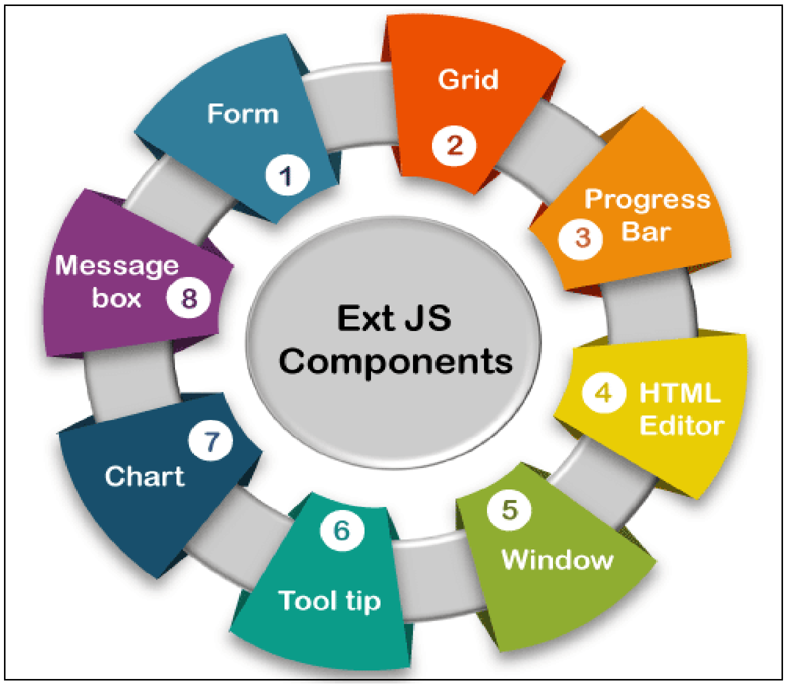


Figure.2.1 Ext.Js Components

The different types of Components and description are shown in Table.2.2.

Sr. No.	Component	Description
1.	**Form**	The form helps us to obtain the data from the user.
2.	**Grid**	It is used to display the data in the form of a table.
3.	**Chart**	It is used to display the data in a pictorial manner.
4.	**Message Box**	This component is used to display the data in an alert box form.
5.	**Window**	The **Window** component helps us to create the window. It is always popped up when an event occurs.
6.	**Tool tip**	It is used to display the data during the event occurring.
7.	**HTML editor**	The editor is used to style the data input by the user. It should be color, size, and font.
8.	**Progress Bar**	This bar always shows the progress of the backend functions.

Table 2.2 Different types of Components

Alert boxes

Different type of alert boxes in Ext JS are

- Ext.MessageBox.alert();
- Ext.MessageBox.confirm();
- Ext.MessageBox.wait();

- Ext.MessageBox.prompt();
- Ext.MessageBox.show();

Getting Started with Ext Js Applications

```
<!DOCTYPE html>
<html>
<head>
<link href = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/classic/theme-classic/resources/theme-classic-all.css"
rel = "stylesheet" />
<script type = "text/javascript"
 src = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/ext-all.js"></script> <script type = "text/javascript">
</script>
</head>
<body>
<div id="helloWorldPanel"></div>
</body>
</html>
Ext.onReady(function() {
Ext.create('Ext.Panel', {
renderTo: 'helloWorldPanel',
height: 500,
width: 500,
title: 'Hello world',
html: 'Welcome to the world of Ext'
});
});
```

Classes in Ext Js

- Ext JS is a JavaScript framework having functionalities of object-oriented programming.
- Ext provides more than 300 classes, which we can use for various functionalities.
- Ext.define() is used for defining the classes in Ext JS.
- An Ext class includes approx. 59 properties and 78 methods.
- Some of the essential methods are **Ext.apply, Ext.create, Ext.define, Ext.getCmp, Ext.override, and Ext.application etc.**

- **Syntax**

  Ext.define(class name, class members/properties, callback function);

- **Class Name:** It is the class name that is given by the user depending upon the application structure.
- **Class Member/Properties:** Class member/properties are used to determine the class behavior.
- **Callback Function:** It is a function that is invoked when the class is loaded. It is an optional function to use.

Ext.Base

- Ext.Base is root of all classes created with Ext.define.
- All the classes in Ext JS inherit from Ext.Base

```
Ext.define('Student',
{
   name : 'unnamed',
   getName : function(){
      return "Student name is" + this.name;
   }
}, function(){
   alert('Student object created');
});
```

Create an Object of a Class

- JavaScript allows us to create an object using new keyword.
- Sencha recommends to use Ext.create() method to create an object of a class which is created using Ext.define() method.

```
var studentObj = Ext.create('Student');
  studentObj.getName();
```

Example

```
<!DOCTYPE html>
<html>
<head>
<link href = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/classic/theme-classic/resources/theme-classic-all.css"
rel = "stylesheet" />
<script type = "text/javascript"
 src = "https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/ext-all.js"></script> <script type = "text/javascript">
  Ext.define('Student',
{
    name : ‘suji',
    getName : function(){
        return "Student name is" + this.name;
    }
}, function(){
    alert('Student object created');
     var studentObj = Ext.create('Student');
alert(studentObj.getName());
});
</script>
</head>
<body>
<div id="helloWorldPanel"></div>
</body>
</html>
```

Define Constructor

- A **constructor** is a special function that gets called when a Class is instantiated.

Example

```
 Ext.define('Student', {
    name : 'unnamed',
    getName : function(){
        return 'Student name is ' + this.name;
    },
    constructor : function(studentName){
        if(studentName)
            this.name = studentName;
    }
});
```

- Now you can create an object by passing parameter to the constructor:
- var studentObj = Ext.create('Student','XYZ');
- studentObj.getName();

Declare Private Members in Class

- In JavaScript, there are two types of object fields.

- **Public:**
- accessible from anywhere.
- They comprise the external interface.
- Until now we were only using public properties and methods.
- **Private:**
- accessible only from inside the class.
- These are for the internal interface.

Example

```
Ext.define('Student', function(){
    var name = 'abc';
    return {
        constructor : function(name){
            this.name = name;
        },
        getName : function(){
            alert('Student name is' + this.name);
        }
    };
});
//create an object of Student class
var studentObj = Ext.create('Student','XYZ');
studentObj.getName();
```

Declare Static Members in Class

- The static members in an Ext JS class can be accessed without creating an object using Ext.create() method.
- It can be accessed using class name with dot notation same as other programming languages.
- Declare static members in Ext JS class using 'statics’ parameter

Example

```
Ext.define('Student',
{
    name : 'abc',
    getName : function(){
        alert('Student name is ' + this.name);
    },
    constructor : function(studentName){
        if(studentName)
            this.name = studentName;
    },
  statics :
    {
        getSchoolName : function(){
            return "XYZ";
                                        }
                                    }
                                });
                                //call static method
                                alert(Student.getSchoolName());
```

Inheritance in Ext JS

- Ext JS supports object-oriented programming concepts such as class, inheritance, polymorphism etc.
- We can inherit a new class from existing class using extend keyword while defining a new class in Ext JS.

Example

```
Ext.define('Person',
{
    name : 'abc',
     getName : function(){
        alert("My name is " + this.name);
    },
    constructor : function(name){
        if(name){
            this.name = name;
        }
    }
            });
```

Example

```
Ext.define('Student',
{
    extend : 'Person',
    schoolName : 'sathyabama',
    constructor : function(name, schoolName){
        this.schoolName = schoolName;
        //call parent class constructor
        this.callParent(arguments);
    },
    getSchoolName : function(){
        alert("My school name is " + this.schoolName);
    }
});
var newStudent = new Student('Queen', 'American School');
```

```
newStudent.getName();
        newStudent.getSchoolName();
```

Mixins

- Mixins introduced since Ext JS 4.
- Mixins allows us to use functions of one class as a function of another class without inheritance.
- Mixins can be defined using mixins keyword
- The value of a property will be name of the class where the method is defined.

Example

```
Ext.define('Person', {
   name: 'abc',
   constructor: function(name) {
      if (name) {
         this.name = name;
      }
   },
   getName: function() {
      alert("My name is " + this.name);
   },
   eat: function(foodType) {
      alert("I like " + foodType);
   }
});
Ext.define('Student', {
   schoolName: 'sathyabama',
   constructor: function(schoolName) {
      this.schoolName = schoolName
```

```
    },
    mixins: {
        eat: 'Person'
    },
    getSchoolName: function() {
        alert("I am a student of " + this.schoolName);
    }
});
var studentObj = new Ext.create('Student', 'XYZ');
studentObj.eat('Sandwich');
        alert(studentObj.getSchoolName());
```

Config

- Ext JS has a feature called config.
- The config allows you to declare public properties with default values which will be completely encapsulated from other class members.
- Properties which are declared via config, will have get() and set().
- The config properties can be defined using config keyword.
- Initconfig- This will create get and set methods for all the config properties.
    - in the constructor in order to initialize getters and setters.
- The get method returns a value of a config property.
- Set method is used to assign a new value to a config property.
- The name of the **get method** will start from get and suffix with property name
    - get<config property name>()
- **Set method** will be named as
    - set<config property name>().
- The suffix in get and set method names will start from capital character.

Example

```
Ext.define('Student', {
    config :
    {
        name : ‘xxxx’,
        schoolName : ‘abc'
    },
    constructor : function(config){
        this.initConfig(config);
    }
});
var newStudent = Ext.create('Student', { name: 'XYZ', schoolName: 'ABC
School' });
newStudent.getName();//output: XYZ
newStudent.getSchoolName();//output: ABC School
newStudent.setName('John');
newStudent.setSchoolName('New School');
alert(newStudent.getName());//output: John
alert(newStudent.getSchoolName());
```

Description

- getName(), setName() and getSchoolName(), setSchoolName() methods in Student object.
- It was automatically created for config properties by Ext JS API.
- Cannot assign config property value directly same as normal class property.

```
newStudent.name = ‘bnn'; //Not valid.
newStudent.setName(‘bvbb');//Valid
```

Custom Setters

- get and set methods are created for config property automatically.
- Custom setters allow you to add extra logic.
- There are two custom setters: **apply and update.**

- The **apply()** method for config property allows us to add some extra logic before it assigns the value to config property.
- The name of apply method must start with apply with config property name as suffix in CamelCase.

  Example: applyName

Custom Setters -Update()

- The update() method executes after the configuration property value has been assigned.
- The name of update method must start with update with config property name as suffix in CamelCase.
- It has two parameters, newValue and oldValue.

  Example: updateName : function(newValue, oldValue)

Example

```
Ext.define('Student',{
   config :
   {
      name : 'unnamed',
      schoolName : 'Unknown'
   },
   constructor : function(config){
      this.initConfig(config);
   },
   applyName : function(name){
      return Ext.String.capitalize(name);
   },
   updateName : function(newValue, oldValue){
      alert('New value: ' + newValue + ', Old value: ' + oldValue);
   }
});
```

```
var newStudent = Ext.create('Student', {name : 'xyz', schoolName : 'ABC School'});

newStudent.setName('john');
```

Alternate ClassName

- Defines alternate names for the class.

Example

```
Ext.define('Developer',
{
   alternateClassName: ['Cc'],
   code: function(msg) {
      alert('Typing... ' + msg);
   }
});
var obj = Ext.create('Developer');
obj.code('hi');
var rms = Ext.create('Cc');
         rms.code('hello');
```

requires

- List of classes that have to be loaded when a class is invoked.
- **Example**

```
define('Mother', {
   requires: ['Child'],
   giveBirth: function() {
      // we can be sure that child class is available.
```

```
        return new Child();
    }
        });
```

Uses

- List of optional classes to load together with this class.
- **Example**

```
Ext.define('Mother', {
    uses: ['Child'],
    giveBirth: function() {
        // This code might, or might not work:
        // return new Child();
        // Instead use Ext.create() to load the class at the spot if not loaded already:
        return Ext.create('Child');
    }
        });
```

Ext.ClassManager

- Ext.ClassManager manages all classes and handles mapping from string class name to actual class objects throughout the whole framework.
    - Ext.define
    - create
    - Ext.widget
    - Ext.getClass
    - Ext.getClassName

Methods

- **Get-**Retrieve a class by its name.

- **getByAlias-** Get a reference to the class by its alias.
- **getByConfig-**Get a component class name from a config object.
- **getClass** (object)-Get the class of the provided object.
- **getDisplayName** (object)-Returns the displayName property or className or object.
- **getName** (object)-Get the name of the class by its reference or its instance.
- **isCreated** (className)
    - Checks if a class has already been created.

Object

- **defineProperties** (obj, props)-Defines new or modifies existing properties directly on an object, returning the object.
- **freeze** (obj) -Nothing can be added to or removed from the properties set of a frozen object. Any attempt to do so will fail
- **getOwnPropertyNames** (obj)-Returns an array of all properties
- **isFrozen** (obj)-Determines if an object is frozen.
- **isSealed** (obj)-Determines if an object is sealed.
- **seal** (obj)-Seals an object, preventing new properties from being added to it and marking all existing properties as non-config
- **preventExtensions** (obj)-Prevents new properties from ever being added to an object

What is the difference between freeze and seal in JavaScript ?

- **Object.seal()** allows changes to the existing properties of an object whereas **Object.freeze()** does not allow so.
- **Object.freeze()** makes an object immune to everything even little changes cannot be made.
- **Object.seal()** prevents from deletion of existing properties but cannot prevent them from external changes.
- **Important classes:** The different classes are discussed in Table 2.3.

Class	Description
Ext	The Ext namespace (global object) encapsulates all classes, singletons, and utility methods provided by Sencha libraries.
Ext.Base	The root of all classes created with Ext.define. Ext.Base is the building block of all Ext classes. All classes in Ext inherit from Ext.Base. All prototype and static members of this class are inherited by all other classes.
Ext.ClassManager	Ext.ClassManager manages all classes and handles mapping from string class name to actual class objects throughout the whole framework.

Ext.Loader	Ext.Loader is the heart of the new dynamic dependency loading capability in Ext JS 4+. It is most commonly used via the Ext.require shorthand.

Table 2.3 Important Classes

Ext.is Class

- This class checks the platform you are using, whether it is a phone or a desktop, a mac or Windows operating system.
- **Ext.is.Platforms-**This function returns the platform available for this version.
- **Ext.is.Android**-This function will return true, if you are using Android operating system, else it returns false
- **Ext.is.Desktop-**This function will return true, if you are using a desktop for the application, else it returns false
- **Ext.is.Phone-**This function will return true, if you are using a mobile, else it returns false.
- **Ext.is.iPhone-**This function will return true if you are using iPhone, else it returns false.
- **Ext.is.iPod-**This function will return true, if you are using iPod, else it returns false.
- **Ext.is.iPad-**This function will return true, if you are using an iPad, else it returns false.
- **Ext.is.Windows-**This function will return true, if you are using Windows operating system, else it returns false.

Ext.supports Class

- This class provides information if the feature is supported by the current environment of the browser/device or not.
- **Ext.supports.History-** It checks if the device supports HTML 5 history as window.history or not.
- **Ext.supports.GeoLocation-**It checks if the device supports geolocation method or not.
- **Ext.supports.Svg-**It checks if the device supports HTML 5 feature scalable vector graphics (svg) method or not.

Ext.String Class

- Ext.String class has various methods to work with string data. The most used methods are encoding decoding, trim, toggle, urlAppend, etc.
- **Ext.String.trim-**This function is to trim the unwanted space in the string.
- **Ext.String.urlAppend-**This method is used to append a value in the URL string.

String

- It is a global object that may be used to construct String instances.
- String objects may be created by calling the constructor new String().
- **charAt** (index)-Returns the character at the specified index.
- **concat** (strings)-Combines combines the text from one or more strings and returns a new string.

```
var hello = "Hello, ";
alert(hello.concat("Kevin", " have a nice day."));
```

Functions

- **Ext.each**
  - Ext.each applies a function to each member of an array.
  - It’s basically a more convenient form of a for loop.

Example

```
var countries = ['Vietnam', 'Singapore', 'United States', 'Russia'];
Ext.Array.each(countries, function(name, index, countriesItSelf) {
```

```
            alert(countries[index])
        });
```

Ext.iterate

- Ext.iterate is like Ext.each for non-array objects.
- **Example**

```
 ships= {'Bill': 'wonderful', 'Laura': 'great'};
Ext.iterate(ships, function(key, value) {
alert(key + "'s ship is the " + value);
});
```

Ext.pluck

- Ext.pluck grabs the specified property from an array of objects:

Example:

```
var animals = [
{name: 'Ed', species: 'Unknown'},
{name: 'Bumble', species: 'Cat'},
{name: 'Triumph', species: 'Insult Dog'}
];
alert(Ext.pluck(animals, 'species'));
alert(Ext.pluck(animals, 'name'));
```

Ext.invoke

- Invoke allows a function to be applied to all members of an array, and returns the results.
- **Example:**

```
var animal=[
{name: 'Ed', species: 'Unknown'},
{name: 'Bumble', species: 'Cat'},

{name: 'Triumph', species: 'Insult Dog'}

];

var describeAnimal = function(animal) {

return String.format("{0} is a {1}", animal.name, animal.species);

}

var describedAnimals = Ext.invoke(animals, describeAnimal);

alert(describedAnimals); // ['Ed is a Unknown', 'Bumble is a Cat', 'Triumph is a
Insult Dog'];
```

Ext.partition

- Ext.Partition splits an array into two sets based on a function you provide:
- **Example**

```
var trees = [

{name: 'Oak',height: 20},

{name: 'Willow', height: 10},

{name: 'Cactus', height: 5} ];

var isTall = function(tree) {return tree.height > 15};

alert(Ext.partition(trees, isTall));
```

Math functions

- **Example**
  - var numbers = [1, 2, 3, 4, 5];
  - alert(Ext.min(numbers)); //1
  - alert(Ext.max(numbers)); //5
  - alert(Ext.sum(numbers)); //15
  - alert(Ext.mean(numbers)); //3

Date

- The following example converts the date object to a numerical value using number as a function .

- **Example**

```
d = new Date("December 17, 1995 03:24:00");
alert(Number(d));
```

Array

- An array is a JavaScript object.
- **Creating an Array**

```
var msgArray = new Array();
```

- **Example**

```
var msgArray = new Array();
msgArray[0] = "Hello";
msgArray[99] = "world";
if (msgArray.length == 100)
alert("The length is 100.");
```

Accessing array elements

- Array elements are nothing less than object properties, so they are accessed as such.
- **Method 1:**

```
var myArray = new Array("Wind", "Rain", "Fire");
alert(myArray[0]); // "Wind”
```

- **Method 2:**

```
myArray[02]
```

Methods of Array

- **indexOf** (searchElement, [fromIndex])
    - Returns the first index at which a given element can be found in the array, or -1 if it is not present.

Example

```
var array = [2, 5, 9];
var index = array.indexOf(2);
alert(index);// index is 0
index = array.indexOf(7);
// index is -1
```

- **join** (separator)-Joins all elements of an array into a string.
- **Example**

```
var a = new Array("Wind","Rain","Fire");
var myVar1 = a.join();
var myVar2 = a.join(", ");
var myVar3 = a.join(" + ");
```

- **lastIndexOf** (searchElement, [fromIndex])
    - Returns the last index at which a given element can be found in the array, or -1 if it is not present.
        - » var array = [2, 5, 9, 2];
        - » var index = array.lastIndexOf(2);
        - » // index is 3
- **pop -**The pop method removes the last element from an array and returns that value to the caller.

```
var myObj = ['angel', 'sonal', 'manal', 'somu'];
var popped = myObj.pop();
alert(popped); // Alerts ‘somu'
```

- **push** (elements)-Adds one or more elements to the end of an array and returns the new length of the array.

```
var sports = ['soccer', 'baseball'];
sports.push(['football', 'swimming']);
alert(sports);
```

- **Reverse-**Reverses the order of the elements of an array

```
var myArray = ["one", "two", "three"];

alert(myArray.reverse());
```

- **sort-**Sorts the elements of an array.

```
var numbers = [4, 2, 5, 1, 3];

alert(numbers.sort());
```

Methods of String

- replace (pattern, replacement)
- search (regexp)
- toLocaleLowerCase()
- var upperText="SENCHA";
- document.write(upperText.toLocaleLowerCase());
- toLocaleUpperCase
- Trim()
- To UpperCase()

Ext.Window

- Ext JS take configuration parameters, many of which can be changed at runtime.
- Ext.Window , has a ‘title’ configuration, which takes the default value of ‘Window Title’.
- 4 methods for free – getTitle, setTitle, resetTitle and applyTitle.
- **getTitle** – returns the current title
- **setTitle** – sets the title to a new value
- **resetTitle** – reverts the title to its default value (‘Window Title’)
- **applyTitle** – this is a template method that you can choose to define. It is called whenever setTitle is called.

```
Ext.define('Ext.Window', {

//..as above,

config: {

title: 'Window Title'

},
applyTitle: function(newTitle) {
this.titleEl.update(newTitle);
}
});
```

Ext JS MVC framework

- **Model**: It is the collection of fields.
- **Store**: It is the collection of data. Each store is associated with a model. ...
- **View**: It is any type of UI component like grid, chart etc.
- **Controllers:** In controllers we put all the codes that makes our app work.

Model-View-Controller

- Model-View-Controller (MVC) is an architectural pattern for writing software.
- It divides the user interface of an application into three distinct parts, helping to organize the codebase into logical representations of information depending upon their function shown in Figure 2.2.

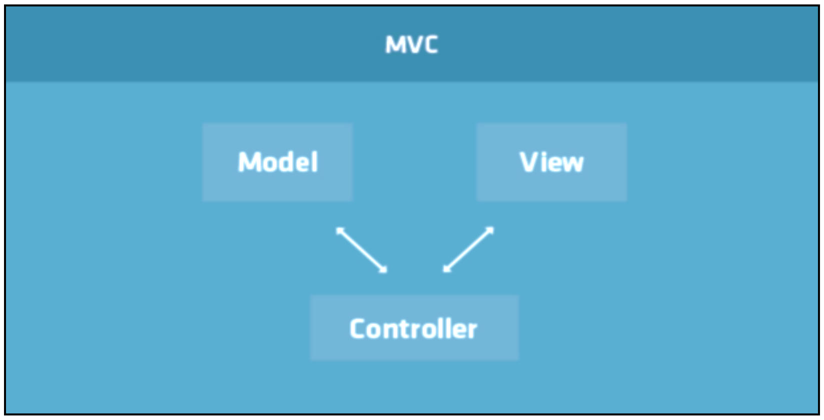


Figure 2.2 MVC Architecture Diagram

- The *Model* describes a common format for the data being used in the application. It may also contain business rules, validation logic, and various other functions.
- The *View* represents the data to the user. Multiple views may display the same data in different ways (e.g. charts versus grids).

- The *Controller* is the central piece of an MVC application. It listens for events in the application and delegates commands between the *Model* and the *View*.

Advantages of MVC

- Easy code maintenance , easy to extend
- MVC Model component can be tested separately from the user
- Easier support for new type of clients
- Development of the various components can be performed parallelly.
- It helps you to avoid complexity by dividing an application into the three units. Model, view, and controller
- It only uses a Front Controller pattern which process web application requests through a single controller.

Disadvantages of using MVC

- Difficult to read, change, to unit test, and reuse this model
- No formal validation support
- Increased complexity
- The difficulty of using MVC with the modern user interface
- There is a need for multiple programmers to conduct parallel programming.
- Knowledge of multiple technologies is required.
- Maintenance of lots of codes in Controller

Model-View-ViewModel

- Model-View-ViewModel (MVVM) is another architectural pattern for writing software that is largely based on the MVC pattern.
- The key difference between MVC and MVVM is that MVVM features an abstraction of a *View* (the *ViewModel*) which manages the changes between a *Model*'s data and the *View*'s representation of that data (i.e. data bindings) — something which typically is cumbersome to manage in traditional MVC applications.

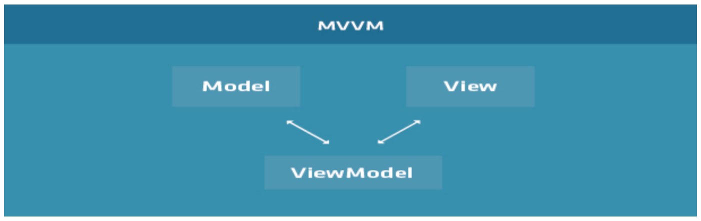


Figure 2.3 MVVM Architecture Diagram

Elements of the MVVM

- The Model describes a common format for the data being used in the application, just as in the classic MVC pattern shown in Figure 2.3.
- The View represents the data to the user, just as in the classic MVC pattern.
- The ViewModel is an abstraction of the view that mediates changes between the View and an associated Model. In the MVC pattern, this would have been the responsibility of a specialized Controller, but in MVVM, the ViewModel directly manages the data bindings and formulas used by the View in question.

Javascript in web browser

- **Internet Explorer**
  - On web browser menu click "Tools" menu and select "Internet Options".
  - In the "Internet Options" window select the "Security" tab.
  - On the "Security" tab click on the "Custom level..." button.
  - When the "Security Settings - Internet Zone" dialog window opens, look for the "Scripting" section.
  - In the "Active Scripting" item select "Enable".
  - When the "Warning!" window pops out asking "Are you sure you want to change the settings for this zone?" select "Yes".
  - In the "Internet Options" window click on the "OK" button to close it.
  - Click on the "Refresh" button of the web browser to refresh the page.
- Internet Explorer < 9
  - On web browser menu click "Tools" and select "Internet Options".
  - In the "Internet Options" window select the "Security" tab.
  - On the "Security" tab click on the "Custom level..." button.
  - When the "Security Settings - Internet Zone" dialog window opens, look for the "Scripting" section.

  - In the "Active Scripting" item select "Enable".
  - When the "Warning!" window pops out asking "Are you sure you want to change the settings for this zone?" select "Yes".
  - In the "Internet Options" window click on the"OK" button to close it.
  - Click on the "Refresh" button of the web browser to refresh the page.
- Mozilla Firefox < 4
  - On the web browser menu click "Tools" and select "Options".
  - In the "Options" window select the "Content" tab.
  - Mark the "Enable JavaScript" checkbox.
  - In the opened "Options" window click on the "OK" button to close it.
  - Click on the "Reload current page" button of the web browser to refresh the page.
- Google Chrome
  - On the web browser menu click on the "Customize and control Google Chrome" and select "Options".
  - In the "Google Chrome Options" tab select the "Under the Hood" menu item.
  - In the "Privacy" section click "Content settings..." button.
  - In the "Content settings" window go to "JavaScript" section and select "Allow all sites to run JavaScript (recommended)".
  - Close the "Google Chrome Options" tab.
  - Click on the "Reload this page" button of the web browser to refresh the page.
- Google Chrome < 10
  - On the web browser menu click on the "Customize and control Google Chrome" and select "Options".
  - In the "Google Chrome Options" window select the "Under the Hood" tab.
  - In the "Privacy" section click "Content settings..." button.
  - In the "Content settings" window select the "JavaScript" tab.
  - In the "JavaScript" tab select "Allow all sites to run JavaScript (recommended)".
  - Close the "Content Setting" window.
  - Close the "Google Chrome Options" window.
  - Click on the "Reload this page" button of the web browser to refresh the page.

- Apple Safari
  - On the web browser menu click on the "Edit" and select "Preferences".
  - In the "Preferences" window select the "Security" tab.
  - In the "Security" tab section "Web content" mark the "Enable JavaScript" checkbox.
  - Click on the "Reload the current page" button of the web browser to refresh the page.
- Opera
  - 1. a) Click on "Menu", over mouse on the "Settings" then over mouse on the "Quick preferences" and mark the "Enable JavaScript" checkbox.
  - 1. b) If "Menu bar" is shown click on the "Tools", hover mouse on the "Quick preferences" and mark the "Enable JavaScript" checkbox.
- Opera < v. 10
  - On the web browser menu click "Tools" and select "Preferences".
  - In the "Preferences" window select the "Advanced" tab.
  - On the "Advanced" tab click on "Content" menu item.
  - Mark the "Enable JavaScript" checkbox.
  - In the opened "Preferences" window click on the "OK" button to close it.
  - Click on the "Reload" button of the web browser to refresh the page.

REFERENCES

1. David Flanagan, " JavaScript: The Definitive Guide " , 6th Edition, O'Reilly Media, Inc, March 2011.

2. Alessio Malizia , Ext JS Documentation Site " Mobile 3D Graphics ", Springer,2006.

3. Colin Ramsay , Shea Frederick , Steve Cutter' Blades, " Learning Ext JS ", Packt Publishing , 2008.

4. Jesus Garcia, " Ext JS in Action ", Manning Publications , 2010.