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 – III - Rich Internet Applications – SCS1401

II. Introduction

Fundamental Classes - Event Handling - Panels and Grids - Layouts and Widgets - Working with data.

Components in Ext JS

- UI is made up of one or more widgets called Components.
- Ext JS provides different types of components out of the box with complex functionalities, which you can use in your application such as Viewport, panel, container, grid, textfield, combobox, numberfield etc. .
- Component is a group of DOM elements with complex functionality.
- Ext JS UI components are derived from the Ext.Component class, which in-turn is derived from the Ext.Base class.
- All the Ext JS components inherit from Ext.Component class.
- Ext.Component class is derived from the Ext.Base class.
- All the components in Ext JS are registered with Ext.ComponentManager class on creation.
- It can be accessed by id using Ext.getCmp() method.
- Example: Ext.getCmp('myComponent')

Example

```
Ext.onReady(function () {
   Ext.create('Ext.Component', {
      id:'myComponent',
      renderTo: Ext.getBody(),   //render method to render to body
      html:'Hello World!'
   });
});
```

Component Life Cycle

- Each component in Ext JS passes through following three stages is shown in Table 3.1.

Stage	Description
Initialization	Registering the component with Ext.ComponentManager and deciding if a component will be rendered.
Rendering	Creating the DOM for the component and adding it to the DOM hierarchy with the events, CSS, etc..
Destruction	Removing the events, the DOM object and unregistering the component from the Ext.ComponentManager

Table 3.1 Stages of Component Life Cycle

Ext JS UI Components

Every Component has a symbolic name called 'xtype'.

Example: Ext.panel.Panel has xtype : 'panel' is discussed in Table 3.2.

Component Name	Class Name	Xtype
ComboBox	Ext.form.field.ComboBox	combobox or combo
Radio Button	Ext.form.field.Radio	radio or radiofield
Checkbox	Ext.form.field.Checkbox	checkbox or checkboxfield
TextField	Ext.form.field.Text	Textfield
Label	Ext.form.Label	Label
Button	Ext.button.Button	Button
DateField	Ext.form.field.Date	Datefield

File Upload	Ext.form.field.File	filefield, fileuploadfield
Hidden Field	Ext.form.field.Hidden	Hidden
Number Field	Ext.form.field.Number	Numberfield
Spinner	Ext.form.field.Spinner	Spinnerfield
Text Area	Ext.form.field.TextArea	Textarea
Time Field	Ext.form.field.Time	Timefield
Trigger	Ext.form.field.Trigger	triggerfield, trigger
Chart	Ext.chart.CartesianChart	Chart
Html Editor	Ext.form.field.HtmlEditor	Htmleditor

Table 3.2 Ext JS UI Components

Combo box

```
var states = Ext.create('Ext.data.Store’,
{
  fields: ['abbr', 'name'],
 data : [        {"abbr":"AL", "name":“Sonal"},
{"abbr":"AK", "name":“Monal"},
{"abbr":"AZ", "name":“Konal"}     ]
});
Ext.create('Ext.form.ComboBox’,
{
fieldLabel: 'Choose State',
store: states,
 queryMode: 'local',
displayField: 'name',
valueField: 'abbr',
renderTo: Ext.getBody();
});
```

Text

- The Text field has a useful set of validations and able to input the data.

Example

```
Ext.create('Ext.form.Panel’,
{    title: 'Contact Info',
 width: 300,
  bodyPadding: 10,
   renderTo: Ext.getBody(),
   items: [{
  xtype: 'textfield',
  name: 'name',
 fieldLabel: 'Name',
 allowBlank: false
},
{
 xtype: 'textfield',
  name: 'email',
 fieldLabel: 'Email Address',
 vtype: 'email’
  }]});
```

Label

```
Ext.create('Ext.form.Panel',
{
title: 'Field with Label',
width: 400,
bodyPadding: 10,
renderTo: Ext.getBody(),
```

```
layout: {        type: 'hbox',
align: 'middle'
},
 items: [{
 xtype: 'textfield',
hideLabel: true,
 flex: 1
},]});
```

Button

```
Ext.create('Ext.Button',
{
 text: 'Click me',
 renderTo: Ext.getBody(),
 handler: function()
{
alert('You clicked the button!');
}});
```

Date

```
Ext.create('Ext.form.Panel',
{    renderTo: Ext.getBody(),
     width: 300,
     bodyPadding: 10,
     title: 'Dates',
     items: [{
       xtype: 'datefield',
      fieldLabel: 'From',
```

```
        name: 'from_date',
        maxValue: new Date()
    }]});
```

Number

```
        Ext.create('Ext.form.Panel’,
        {
        title: 'Numberings',
        width: 300,
        bodyPadding: 10,
        renderTo: Ext.getBody(),
        items: [{
        xtype: 'numberfield’,
        name: 'bottles’,
        fieldLabel: 'Number starts’,
        value: 99,
        maxValue: 99,
        minValue: 0
        }],});
```

Text Area

```
        Ext.create('Ext.form.FormPanel',
        {
          title      : 'My Document',
          width      : 400,
          bodyPadding: 10,
          renderTo   : Ext.getBody(),
          items: [{
            xtype     : 'textareafield',
            name      : 'message',
```

```
fieldLabel: 'Message',
anchor    : '100%'
}]});
```

HTML Editor

```
Ext.tip.QuickTipManager.init();
Ext.create('Ext.form.HtmlEditor',
{
width: 580,
height: 250,
renderTo: Ext.getBody()
});
```

Charts

- Charts are used to represent data in pictorial format.
- The different charts provided by Ext JS −
    - Pie Chart
    - Line Chart
    - Bar Chart
    - Area Chart

Pie Chart

- Syntax

```
Ext.create('Ext.chart.PolarChart', {
  series: [{
    Type: 'pie'
    ..
  }]
  render and other properties.
});
```

Container

- Ext JS includes components that can contain other components are called **container**.
- Ext.container.Container is a base class for all the containers in Ext JS.
- Add different types of Ext JS components into a container using **items** config property or **add()** method.
- Container can also include another container.
- We can add or remove the components from the container and from its child elements.
- Ext.container.Container is the base class for all the containers in Ext JS.

Example

```
Ext.create('Ext.container.Container’,
 {
    layout:
   {
          type: 'hbox'
  },
     width: 400,
      renderTo: Ext.getBody(),
      border: 1,
      defaults: {
      labelWidth: 80, xtype: 'datefield',
  },
  items: [{
       xtype: 'datefield',
        name: 'Dates',
        fieldLabel: 'Dates’
        }]});
```

Components inside container

```
var component1 = Ext.create('Ext.Component', {
  html:'First Component'
```

```
});
Ext.create('Ext.container.Container', {
  renderTo: Ext.getBody(),
  items: [component1]
});
```

Container inside container

```
var container = Ext.create('Ext.container.Container', {
  items: [component3, component4]
});
Ext.create('Ext.container.Container', {
  renderTo: Ext.getBody(),
  items: [container]
});
```

Types of Container

- Ext.panel.Panel
- Ext.form.Panel
- Ext.tab.Panel
- Ext.container.Viewport

Ext.panel.Panel

- It is the basic container which allows to add items in a normal panel.

```
var childPanel1 = Ext.create('Ext.Panel', {
  html: 'First Panel'
});
Ext.create('Ext.panel.Panel', {
  renderTo: Ext.getBody(),
  width: 100,
  height : 100,
  border : true,
```

```
        frame : true,
        items: [ childPanel1]
});
```

Ext.form.Panel Container

- Form panel provides a standard container for forms.
- Ext.panel.Panel, which automatically creates a BasicForm for to manage any Ext.form.field.Field objects.
- **Example**

```
var child1 = Ext.create('Ext.Panel',{
        html: 'Text field'
      });

Ext.create('Ext.form.Panel', {
        renderTo: Ext.getBody(),
        width: 100,
        height : 100,
        border : true,
        frame : true,
        layout : 'auto',    // auto is one of the layout type.
        items : [child1]
});
```

Ext.tab.Panel

- Tab panel is like a normal panel but has support for card tab panel layout.
- **Example**

```
Ext.create('Ext.tab.Panel', {
        renderTo: Ext.getBody(),
        height: 100,
```

```
    width: 200,

    items: [{
      xtype: 'panel',
      title: 'Tab One',
      html: 'The first tab',

    },{
      // xtype for all Component configurations in a Container
      title: 'Tab Two',
      html: 'The second tab',

    }]
  });
```

Ext.container.Viewport

- Viewport is a container that automatically resizes itself to the size of the whole browser window.

Example

```
var childPanel1= Ext.create('Ext.panel.Panel', {
title: 'Child Panel 1',
html: 'Another Panel'
});
Ext.create('Ext.container.Viewport', {
renderTo: Ext.getBody(),
items: [childPanel1 ]
});
```

Layout

- Layout is the way the elements are arranged in a container.
- It can be horizontal, vertical, or any other.
- Ext JS has a different layout defined in its library but we can always write custom layouts

Types of Layout

- Absolute Layout
- Accordian Layout
- Anchor Layout
- Border Layout
- Auto Layout
- Card Layout
- Column Layout
- Fit Layout
- Table Layout
- vBox Layout
- hBox Layout

Absolute Layout

- This layout allows to position the items using XY coordinates in the container.

  Syntax

  Layout: ‘absolute’

Example

```
Ext.create('Ext.container.Container', {
        renderTo: Ext.getBody(),
        layout: 'absolute' ,
        items: [{
          title: 'Panel 1',
          x: 50,
```

```
            y: 50,
            html: 'Positioned at x:50, y:50',
            width: 500,
            height: 100
        }]
});
```

Accordion Layout

- This layout allows to place all the items in stack fashion (one on top of the other) inside a container.

  Syntax:

  layout:’accordian’

Example

```
renderTo : Ext.getBody(),
layout : 'accordion' ,
width : 600,

items : [{
  title : 'Panel 1',
  html : 'Panel 1 html content'
},{
  title : 'Panel 2',
  html : 'Panel 2 html content'
},{
  title : 'Panel 3',
  html : 'Panel 3 html content'
},{
  title : 'Panel 4',
  html : 'Panel 4 html content'
},{
```

```
            title : 'Panel 5',
            html : 'Panel 5 html content'
        }]
    });
```

Anchor Layout

- This layout gives the privilege to the user to specify the size of each element with respect to the container size.

  Syntax

  layout:’anchor’

Example

```
    Ext.create('Ext.container.Container', {
        renderTo : Ext.getBody(),
        layout : 'anchor' ,
        width : 600,

        items : [{
          title : 'Panel 1',
          html : 'panel 1',
          height : 100,
          anchor : '50%'
        }
        ]
    });
```

Border Layout

- In this layout, various panels are nested and separated by borders.

  Syntax

  Layout:’Border’

Example

```
Ext.create('Ext.panel.Panel', {
        renderTo: Ext.getBody(),
        height: 300,
        width: 600,
        layout:'border',

        defaults: {
          collapsible: true,
          split: true,
          bodyStyle: 'padding:15px'
        },
        items: [{
          title: 'Panel1',
          region:'west',
          html: 'This is Panel 1’
}]
});
```

Auto Layout

- This is the default layout that decides the layout of the elements based on the number of elements.

  Syntax

```
layout:’auto’
```

Example

```
Ext.create('Ext.container.Container', {
        renderTo : Ext.getBody(),
        layout : 'auto',
        width : 600,

        items : [{
```

```
            title: 'First Component',
            html : 'This is First Component'
          },{
            title: 'Second Component',
            html : 'This is Second Component'
          },{
            title: 'Third Component',
            html : 'This is Third Component'
          },{
            title: 'Fourth Component',
            html : 'This is Fourth Component'
          }]
        });
```

Card

- This layout arranges different components in tab fashion.
- Tabs will be displayed on top of the container.
- Every time only one tab is visible and each tab is considered as a different component.

Example

```
    Ext.create('Ext.tab.Panel', {
        renderTo: Ext.getBody(),
        requires: ['Ext.layout.container.Card'],
        xtype: 'layout-cardtabs',
        width: 600,
        height: 200,
        items:[{
          title: 'Tab 1',
          html:   'This is first tab.'
        },{
```

```
            title: 'Tab 2',
            html: 'This is second tab.'
        },{
            title: 'Tab 3',
            html: 'This is third tab.'
        }]
});
```

Column

- This layout is to show multiple columns in the container.
- We can define a fixed or percentage width to the columns.
- The percentage width will be calculated based on the full size of the container.

Example

```
Ext.create('Ext.panel.Panel', {
        renderTo : Ext.getBody(),
        layout : 'column' ,
        xtype: 'layout-column',
        requires: ['Ext.layout.container.Column'],
        width : 600
        items: [{
          title : 'First Component width 30%',
          html : 'This is First Component',
          columnWidth : 0.30
        },{
          title : 'Second Component width 40%',
          html : '<p> This is Second Component </p> <p> Next line for second component </p>',
          columnWidth : 0.40
        }]
    });
```

Fit

- In this layout, the container is filled with a single panel.
- When there is no specific requirement related to the layout, then this layout is used.

Example

```
Ext.create('Ext.container.Container', {
    renderTo : Ext.getBody(),
    layout : {
      type :'fit'
    },
    width : 600,
    defaults: {
      bodyPadding: 15
    },
    items : [{
      title: 'Panel1',
      html : 'This is panel 1'
    },{
      title: 'Panel2',
      html : 'This is panel 2'
    },{
      title: 'Panel3',
      html : 'This is panel 3'
    },{
      title: 'Panel4',
      html : 'This is panel 4'
    }]
});
```

Table Layout

- As the name implies, this layout arranges the components in a container in the HTML table format.

Example

```
Ext.create('Ext.container.Container', {
    renderTo : Ext.getBody(),
    layout : {
      type :'table',
      columns : 3,
      tableAttrs: {
        style: {
          width: '100%'
        }
      }
    },
    width:600,
    height:200,

    items : [{
      title : 'Panel1',
      html : 'This panel has colspan = 2',
      colspan :2
    },{
      title : 'Panel2',
      html : 'This panel has rowspan = 2',
      rowspan: 2
```

```
},{
  title : 'Panel3',
  html : 'This  s panel 3'
},{
  title : 'Panel4',
  html : 'This is panel 4'
},{
  title : 'Panel5',
  html : 'This is panel 5'
}]
});
```

vBox

- This layout allows the element to be distributed in the vertical manner. This is one of the mostly used layout.

Example

```
Ext.create('Ext.panel.Panel', {
    renderTo : Ext.getBody(),
    layout : {
      type :'vbox',
      align: 'stretch'
    },
    requires: ['Ext.layout.container.VBox'],
    xtype: 'layout-vertical-box',
    width : 600,
    height :400,
    frame :true,
    items : [{
      title: 'Panel 1',
```

```
            html : 'Panel with flex 1',
            margin: '0 0 10 0',
            flex : 1
          },{
            title: 'Panel 2',
            html : 'Panel with flex 2',
            margin: '0 0 10 0',
            flex : 2
          }]
});
```

Hbox

- This layout allows the element to be distributed in the horizontal manner.

Example

```
Ext.create('Ext.panel.Panel', {
    renderTo : Ext.getBody(),
    layout : {
      type :'hbox'
    },
    requires: ['Ext.layout.container.HBox'],
    xtype: 'layout-horizontal-box',
    width : 600,
    frame :true,
    items : [{
      title: 'Panel 1',
      html : 'Panel with flex 1',
      flex : 1
    },{
      title: 'Panel 2',
```

```
            html : 'Panel with flex 2',
            flex : 2
        },]
    });
```

Working with Data

- Data package is used for loading and saving all the data in the application.
- Data package has numerous number of classes but the most important classes are −
- Model
- Store
- Proxy

Model

- The base class for model is **Ext.data.Model**.
- It represents an entity in an application.
- It binds the store data to view.
- It has mapping of backend data objects to the view dataIndex.
- The data is fetched with the help of store.

Creating a Model

- For creating a model, we need to extend Ext.data.Model class and we need to define the fields, their name, and mapping.

```
Ext.define('StudentDataModel', {
   extend: 'Ext.data.Model',
   fields: [
      {name: 'name', mapping : 'name'},
      {name: 'age', mapping : 'age'},
      {name: 'marks', mapping : 'marks'}
   ]
});
```

- The name should be the same as the dataIndex, which we declare in the view and the mapping should match the data, either static or dynamic from the database, which is to be fetched using store.

Store

- The base class for store is **Ext.data.Store**.
- It contains the data locally cached, which is to be rendered on 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.
- Store data can be fetched in two ways - static or dynamic.

Static store

- For static store, we will have all the data present in the store as shown in the following code.

```
Ext.create('Ext.data.Store', {
  model: 'StudentDataModel',
  data: [
    { name : "Asha", age : "16", marks : "90" },
    { name : "Vinit", age : "18", marks : "95" },
    { name : "Anand", age : "20", marks : "68" },
    { name : "Niharika", age : "21", marks : "86" },
    { name : "Manali", age : "22", marks : "57" }
  ];
});
```

Dynamic Store

- Dynamic data can be fetched using proxy.
- We can have proxy which can fetch data from Ajax, Rest, and Json.

Proxy

- The base class for proxy is Ext.data.proxy.Proxy.
- Proxy is used by Models and Stores to handle the loading and saving of Model data.
- There are two types of proxies
- Client Proxy

- Server Proxy

Types of proxy

- Client Proxy-Client proxies include Memory and Local Storage using HTML5 local storage.
- Server Proxy-Server proxies handle data from the remote server using Ajax, Json data, and Rest service.

Defining proxies in the server

```
Ext.create('Ext.data.Store', {
  model: 'StudentDataModel',
  proxy : {
    type : 'rest',
    actionMethods : {
      read : 'POST'  // Get or Post type based on requirement
    },
    url : 'restUrlPathOrJsonFilePath', // here we have to include the rest URL path
    // which fetches data from database or Json file path where the data is stored
    reader: {
      type : 'json',  // the type of data which is fetched is of JSON type
      root : 'data'
    },
  }
});
```

Panels

- A panel is a container that has specific functionality and structural components that make it the perfect building block for application-oriented user interfaces.
- We can add toolbars, buttons at the bottom or make the panel collapsible.
- The panels can be assigned easily to another container.
- **Example**

```
var main = new Ext.Panel({
```

```
    title: 'My first panel', //panel's title
    width:250, //panel's width
    height:300, //panel's height
    renderTo: 'frame', //the element where the panel is going to be inserted
    html: 'Nothing important just dummy text' //panel's content
});
```

Types of Panel

Tree Panel

- The Tree Panel Component is one of the most versatile Components in Ext JS and is an excellent tool for displaying heirarchical data in an application.
- Tree Panel extends from the same class as Grid Panel, so all of the benefits of Grid Panels - features, extensions, and plugins can also be used on Tree Panels.
- Things like columns, column resizing, dragging and dropping, renderers, sorting and filtering can be expected to work similarly for both components.

Example

```
Ext.create('Ext.tree.Panel', {
    renderTo: Ext.getBody(),
    title: 'Simple Tree',
    width: 150,
    height: 150,
    root: {
        text: 'Root',
        expanded: true,
        children: [
            {
                text: 'Child 1',
                leaf: true
```

```
            },
            {
                text: 'Child 2',
                leaf: true
            },
            {
                text: 'Child 3',
                expanded: true,
                children: [
                    {
                        text: 'Grandchild',
                        leaf: true
                    }
                ]
            }
        ]
    }
});
```

Grid Panels

- ExtJS Grid, you can create grids which are nothing but tables with rows and columns in it.
- Based on the type of ExtJS Grid you use, you can print and edit the grid contents.
- Edit includes Add/Modify/Delete of a particular column/row in the table.
- ExtJS Grid also takes care of paging when the number of records is huge.

Types of Grid Panels in ExtJS Grid

- **Grid Panel** – Using GridPanel you are allowed to show only the data on the Grid. No other transactions can be done on the Grid.
- **Editor Grid Panel** – Editor Grid Panel extends or inherits from GridPanel. If any of the columns in the Editor Grid Panel has to be editable, then you can configure specific editor for that column.

- **Property Grid** – Property Grid extends or inherits from EditorGridPanel. It is used to manage elements of the Grid as key-value pairs.

Events

- Events are something which get fired when something happens to the class.
- For example, when a button is getting clicked or before/after the element is rendered.

Methods of Writing Events

- Built-in events using listeners
- Attaching events later
- Custom events

Built-in Events Using Listeners

```
Ext.create('Ext.Button', {
        renderTo: Ext.getElementById('helloWorldPanel'),
        text: 'My Button',

        listeners: {
          click: function() {
            Ext.MessageBox.alert('Alert box', 'Button is clicked');
          }
        }
      });
```

Attaching an Event Later

- In the previous method of writing events, we have written events in listeners at the time of creating elements.

  Example

```
button.on('click', function() {
Ext.MessageBox.alert('Alert box', 'Button is clicked');
```

```
});
```

Example

```
var button = Ext.create('Ext.Button’,
{
 renderTo: Ext.getBody(),
text: 'My Button’
 });
button.on('click', function()
{    alert("Success!','Event listener attached by .on’);
});
```

Custom Events

- We can write custom events in Ext JS and fire the events with fireEvent method.
- FireEvent- Events fire whenever something interesting happens to one of your Classes.
- when a Component renders to the screen, Ext JS fires an event after the render completes.

Removing Listeners

- we can add listeners at any time, we can also remove them.
- Use .un function to remove the listener.

Example

```
 doSomething = function() {
    Ext.Msg.alert('listener called');
};
var button = Ext.create('Ext.Button', {
    renderTo: Ext.getBody(),
    text: 'My Button',
    listeners: {
        click: doSomething,
    }
```

```
});
Ext.defer(function() {
   button.un('click', doSomething);
}, 3000);
```

Ext.defer

- Clicking the button in the first 3 seconds yields an alert message.
- However, after 3 seconds the listener is removed so nothing happens

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.