Tabla de contenido

[Chapter 1: Introduction to ExtJS 4](#_Toc373159002)

[What is ExtJS? 4](#_Toc373159003)

[Reviewing ExtJS Documentation 4](#_Toc373159004)

[Code Editor: 5](#_Toc373159005)

[Live Preview 5](#_Toc373159006)

[Reviewing ExtJS Examples 6](#_Toc373159007)

[Introduction ExtJS 4 MVC 7](#_Toc373159008)

[Chapter 2: Creating an ExtJS Project 8](#_Toc373159009)

[Introduction to Sencha CMD 8](#_Toc373159010)

[Explaining the folders structure 9](#_Toc373159011)

[Understanding extjs files and main application files 10](#_Toc373159012)

[ExtJS SDK files 10](#_Toc373159013)

[Index.html 10](#_Toc373159014)

[App.js 11](#_Toc373159015)

[Application.js 11](#_Toc373159016)

[Viewport.js 12](#_Toc373159017)

[Understanding the class system 13](#_Toc373159018)

[Setters and getters 13](#_Toc373159019)

[Apply and Update functions 14](#_Toc373159020)

[Statics and singletons 14](#_Toc373159021)

[Creating the launch function 15](#_Toc373159022)

[Chapter 3: Defining Views 15](#_Toc373159023)

[Understanding the xtype property 15](#_Toc373159024)

[Using the hbox and vbox layouts 15](#_Toc373159025)

[HBox 15](#_Toc373159026)

[VBox 16](#_Toc373159027)

[Using the fit and card layouts 16](#_Toc373159028)

[Fit 16](#_Toc373159029)

[Card 16](#_Toc373159030)

[Using the Column layout 16](#_Toc373159031)

[Chapter 4: Defining Models and Stores 18](#_Toc373159032)

[Defining model fields 18](#_Toc373159033)

[Working with Date fields 19](#_Toc373159034)

[Defining convert functions 20](#_Toc373159035)

[Defining model associations 20](#_Toc373159036)

[Defining model validations 21](#_Toc373159037)

[Chapter 5: Working with forms 22](#_Toc373159038)

[Creating a form 22](#_Toc373159039)

[Understanding the Anchor layout 23](#_Toc373159040)

[Assigning default configuration values 23](#_Toc373159041)

[Using TextFields 24](#_Toc373159042)

[Using ComboBoxes 24](#_Toc373159043)

[Using Radiobuttons 25](#_Toc373159044)

[Using the FieldSet component 26](#_Toc373159045)

[Using the FieldContainer component 27](#_Toc373159046)

[Linking a form to a model 28](#_Toc373159047)

[Updating and validation the model 28](#_Toc373159048)

[Working with VTypes 29](#_Toc373159049)

[Creating custom vtypes 29](#_Toc373159050)

[Chapter 6: Working with grids 30](#_Toc373159051)

[Creating a grid 30](#_Toc373159052)

[Configuring columns 31](#_Toc373159053)

[Understanding column renderers 34](#_Toc373159054)

[Cell editing and row editing 34](#_Toc373159055)

[Cell Editing 34](#_Toc373159056)

[Row Editing 35](#_Toc373159057)

[Chapter 7: Loading data locally and from a server 36](#_Toc373159058)

[Understanding the Proxy class 36](#_Toc373159059)

[Defining a proxy to read a local json file 37](#_Toc373159060)

[Defining a proxy to put data in the LocalStorage 37](#_Toc373159061)

[Defining a proxy to put data in the SessionStorage 37](#_Toc373159062)

[Defining a proxy to read data from a server 37](#_Toc373159063)

[Chapter 8: Defining Controllers 38](#_Toc373159064)

[Creating a Controller 38](#_Toc373159065)

[Defining controllers listeners 39](#_Toc373159066)

[Defining controllers refs 40](#_Toc373159067)

[Chapter 9: Working with templates 41](#_Toc373159068)

[Understanding the XTemplates class 41](#_Toc373159069)

[Implementing conditional processing 41](#_Toc373159070)

[Binding custom javascript methods to XTemplates 42](#_Toc373159071)

[Binding a template to a component 42](#_Toc373159072)

[Explaining the DataView component 43](#_Toc373159073)

[Chapter 10: Creating custom components 44](#_Toc373159074)

[Understanding components life cycle 44](#_Toc373159075)

[Chapter 11: Creating a production build 45](#_Toc373159076)

[Using Sencha CMD to generate a production build 45](#_Toc373159077)

[Chapter 12: Best Practices 46](#_Toc373159078)

# Chapter 1: Introduction to ExtJS

## What is ExtJS?

Sencha Ext JS is the leading standard for business-grade web application development. With over 100 examples, 1000 APIs, hundreds of components, a full documentation suite and built in themes, Ext JS provides the tools necessary to build robust desktop applications. Ext JS also brings a rich data package that allows developers to use a model-view-controller (MVC) architecture when building their app. The MVC leverages features like Big Data Grids enabling an entirely new level of interactivity in web apps.

Ext JS 4 brings a whole new way to build client applications, by introducing the popular model-view-controller pattern to Ext JS. By allowing the separation of data management, logic and interface elements, Ext JS 4 makes it easier for even large development teams to work independently without worrying about stepping on each other's toes. Ext JS 4 ships with a helpful MVC guide to get started.

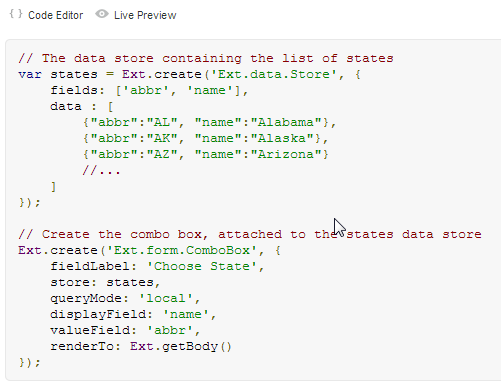
Ext JS 4 lets developers deliver on an incredible variety of browsers and on more operating systems using the same code — over ten years of browsers in one release. On modern browsers, Ext JS 4 utilizes HTML5 features and falls back to alternatives on older browsers. Whether you’re using Ext JS’ built-in UI components, using the Charting package, or theming your application, Ext JS 4 makes it easy to build an app that gives you the power of the web regardless of what browser your customer uses.

## Reviewing ExtJS Documentation

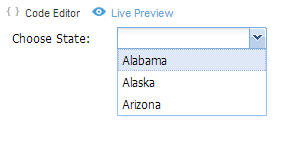
Ext JS and Sencha Touch use the same documentation system, JSDUCK. Therefore all of the things that you learned about in the prior unit regarding the Sencha class system, inheritance, methods, and runtime properties all apply to Ext JS. Ext JS does, however, contain many more visual classes than Sencha Touch, and you should take some time to review them.

Every example has the option to look at the source code and also a live preview in which you can interact with the example.

### Code Editor:

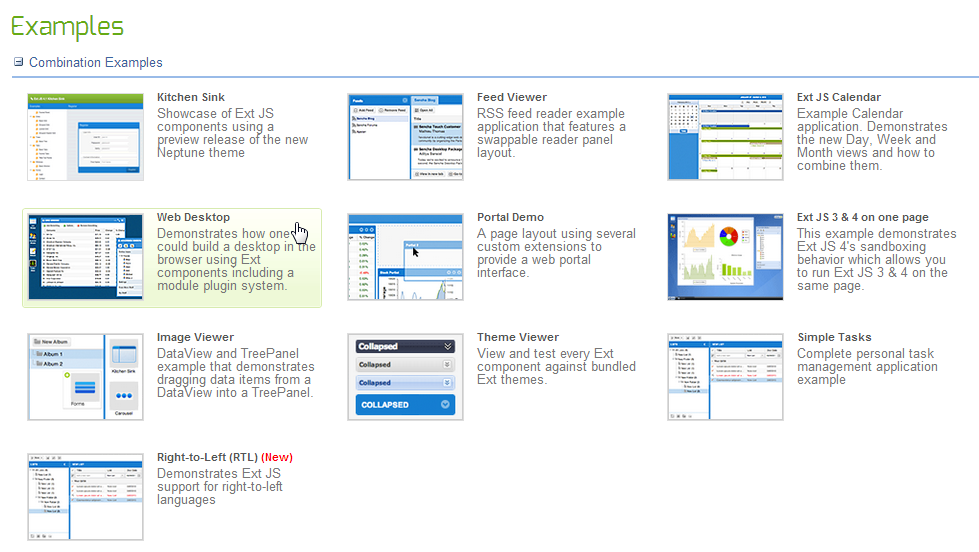


### Live Preview



## Reviewing ExtJS Examples

Ext JS contains some very impressive examples that illustrate the power of the framework.



|  |  |
| --- | --- |
| Example | Description |
| Web Desktop | Illustrates the power of Ext JS by replicating the look and feel of a Windows desktop |
| Kitchen Sink | Similar to the Sencha Touch Kitchen Sink app, it demonstrates how to instantiate basic UI components |
| Keyboard Feed Viewer | Demonstrates some of the accessibility features of Ext, which can be critical for governmental agencies that must produce apps that are accessible to people with disabilities |
| Infinite Grid | Displays a very large dataset in a grid using automatic pagination |
| Multisort Dataview | Uses a dataview with filters and animations to sort graphic representations of data in a store |
| Layout Manager | Demonstrates the Ext 4 layout managers, which are much more robust than those found in Sencha Touch. |

## Introduction ExtJS 4 MVC

Large client side applications have always been hard to write, hard to organize and hard to maintain. They tend to quickly grow out of control as you add more functionality and developers to a project. Ext JS 4 comes with a new application architecture that not only organizes your code but reduces the amount you have to write.

Our application architecture follows an MVC-like pattern with Models and Controllers being introduced for the first time. There are many MVC architectures, most of which are slightly different from one another. Here's how we define ours:

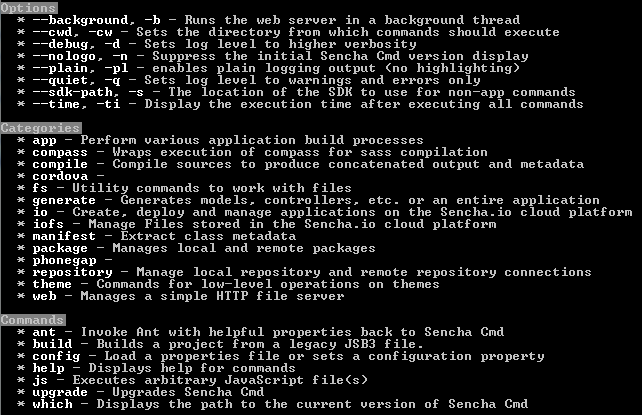
* **Model:** is a collection of fields and their data (e.g. a User model with username and password fields). Models know how to persist themselves through the data package, and can be linked to other models through associations. Models work a lot like the Ext JS 3 Record class, and are normally used with Stores to present data into grids and other components
* **View:** is any type of component - grids, trees and panels are all views.
* **Controllers:** are special places to put all of the code that makes your app work - whether that's rendering views, instantiating Models, or any other app logic.

# Chapter 2: Creating an ExtJS Project

## Introduction to Sencha CMD

Sencha Cmd is a set of command-line utilities that provide the following services:

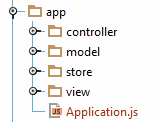
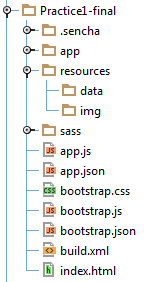
* Generate starter code for your application
* Package your application for deployment on testing and production web servers.
* Package your application as a native iOS and/or Android application that can be uploaded to the iOS App Store / Google Play



## Explaining the folders structure

ExtJS applications follow a unified directory structure that is the same for every app. The code for your application is placed into an 'app' folder, which in turn contains folders for your models, views, controllers and stores as depicted in the illustration.

At the top level there is a single file, index.html, which is the page your application will be hosted from. At the same level there is an 'app' directory, which contains directories for controllers, models, stores and views. The resources folder is used as a container for CSS, images, and other media assets. Sencha Cmd copies the framework into the ext folder in order for you to isolate any framework changes or patches to your specific app. While you can add additional folders to this structure, it is strongly recommended that you keep this base file layout intact.



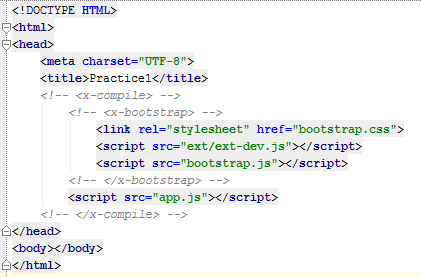
## Understanding extjs files and main application files

### ExtJS SDK files

* ext.js  
  Basic extjs framework and components classes. Minified and obfuscated
* ext-all.js  
  Complete extjs framework and components classes. Minified and obfuscated.
* ext-debug.js  
  Basic. Not minified or obfuscated
* ext-all-debug.js  
  Complete. Not minified or obfuscated
* ext-dev.js  
  Basic with comments and dev errors. Not minified or obfuscated
* ext-all-dev.js  
  Complete with comments and dev errors. Not minified or obfuscated

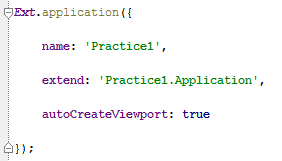
### Index.html

Main css file, extjs sdk and app.js files are included here.



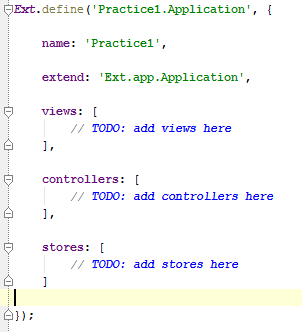
### App.js

Here is where our extjs application is created.



### Application.js

Represents an Ext JS 4 application, which is typically a single page app using a Viewport. Models, stores, controllers and views must be included here.



### Viewport.js

A specialized container representing the viewable application area (the browser viewport).

The Viewport renders itself to the document body, and automatically sizes itself to the size of the browser viewport and manages window resizing. There may only be one Viewport created in a page.

Like any Container, a Viewport will only perform sizing and positioning on its child Components if you configure it with a layout.

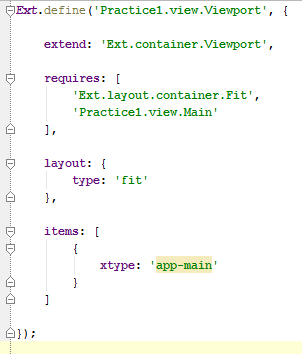
A Common layout used with Viewports is border layout, but if the required layout is simpler, a different layout should be chosen.

For example, to simply make a single child item occupy all available space, use fit layout.

To display one "active" item at full size from a choice of several child items, use card layout.

Inner layouts are available because all Panels added to the Viewport, either through its items, or the add method of any of its child Panels may themselves have a layout.

The Viewport does not provide scrolling, so child Panels within the Viewport should provide for scrolling if needed using the autoScroll config.



## Understanding the class system

* Use Ext.define to define a class configuration
* Use Ext.create to create an instance of a class previously defined

For more detailed information take a look at this link: <http://docs.sencha.com/extjs/4.2.1/#!/guide/class_system>

### Setters and getters

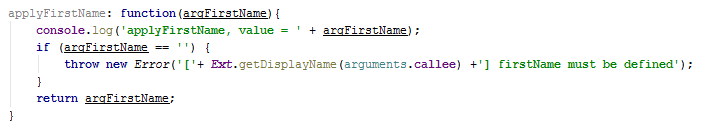
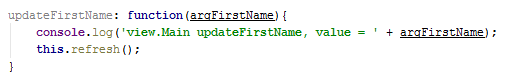
ExtJS framework will generate a getter and a setter for each property that is inside the config object of a class.



As can be seen in this picture, extjs generated a getter for the firstName and lastName properties.

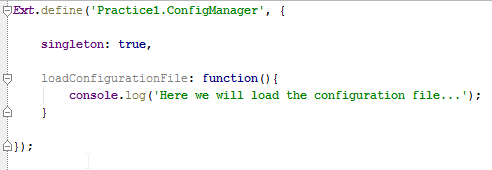
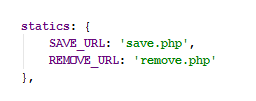
### Apply and Update functions

Users have the option to implement the apply and update functions for every property defined inside the config object of a class.

* Apply: Intercepts and validates a value prior to it being stored in the component property  
  
* Update: Executed after the property has successfully been set  
  

### Statics and singletons

To define static properties for a class or to define a class as a singleton class is really easy:



## Creating the launch function

The load/execution order is as follows:

* app.js is loaded
* All controllers are loaded and their init() methods are called in the order in which they're listed in the Ext.application configuration.
* If the autoCreateViewport property is set to true, your views/Viewport.js file is loaded into memory and an instance is automatically instantiated.
* The launch method is executed.

# Chapter 3: Defining Views

## Understanding the xtype property

xtype is a shorthand way to identify particular components: 'panel' = Ext.Panel, 'textfield' = Ext.form.TextField, etc. When you create a page or a form, you may use these xtypes rather than instantiate objects.

Moreover, creating pages in this manner allows ExtJS to render lazily the page. This is where you see a "performance gain." Instead of creating a large number of components when the app loads, ExtJS renders components when the user needs to see them. Not a big deal if you have one page, but if you exploit tabs or an accordion, many pages are initially hidden and therefore the app will load more quickly.

Furthermore, you may create and register new components creating xtypes of your choosing. ExtJS will similarly render your components lazily.

## Using the hbox and vbox layouts

### HBox

A layout that arranges items horizontally across a Container. This layout optionally divides available horizontal space between child items containing a numeric flex configuration.

This layout may also be used to set the heights of child items by configuring it with the align option.

The possible align values are:

* Stretch = Uses the max possible height for all of its items
* Top, middle and bottom = Arrange items vertically based on their height

### VBox

A layout that arranges items vertically down a Container. This layout optionally divides available vertical space between child items containing a numeric flex configuration.

This layout may also be used to set the widths of child items by configuring it with the align option.

The possible align values are:

* Stetch = Uses the max possible width for all of its items
* Left, center and right = Arrange items horizontally based on their width

## Using the fit and card layouts

### Fit

A layout that must have an only child that will use the width and height of its parent container.

### Card

This layout manages multiple child Components, each fitted to the Container, where only a single child Component can be visible at any given time. This layout style is most commonly used for wizards, tab implementations, etc. This class is intended to be extended or created via the layout:'card' Ext.container.Container.layout config, and should generally not need to be created directly via the new keyword.

The CardLayout's focal method is setActiveItem. Since only one panel is displayed at a time, the only way to move from one Component to the next is by calling setActiveItem, passing the next panel to display (or its id or index). The layout itself does not provide a user interface for handling this navigation, so that functionality must be provided by the developer.

To change the active card of a container, call the setActiveItem method of its layout.

## Using the Column layout

This is the layout style of choice for creating structural layouts in a multi-column format where the width of each column can be specified as a percentage or fixed width, but the height is allowed to vary based on the content. This class is intended to be extended or created via the layout:'column' Ext.container.Container.layout config, and should generally not need to be created directly via the new keyword.

ColumnLayout does not have any direct config options (other than inherited ones), but it does support a specific config property of columnWidth that can be included in the config of any panel added to it. The layout will use the columnWidth (if present) or width of each panel during layout to determine how to size each panel. If width or columnWidth is not specified for a given panel, its width will default to the panel's width (or auto).

The width property is always evaluated as pixels, and must be a number greater than or equal to 1. The columnWidth property is always evaluated as a percentage, and must be a decimal value greater than 0 and less than 1 (e.g., .25).

The basic rules for specifying column widths are pretty simple. The logic makes two passes through the set of contained panels. During the first layout pass, all panels that either have a fixed width or none specified (auto) are skipped, but their widths are subtracted from the overall container width.

During the second pass, all panels with columnWidths are assigned pixel widths in proportion to their percentages based on the total remaining container width. In other words, percentage width panels are designed to fill the space left over by all the fixed-width and/or auto-width panels. Because of this, while you can specify any number of columns with different percentages, the columnWidths must always add up to 1 (or 100%) when added together, otherwise your layout may not render as expected.

# Chapter 4: Defining Models and Stores

## Defining model fields

This is an Array of Field definition objects. A Field definition may simply be the name of the Field, but a Field encapsulates data type, custom conversion of raw data, and a mapping property to specify by name of index, how to extract a field's value from a raw data object, so it is best practice to specify a full set of Field config objects.

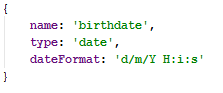


## Working with Date fields

To use a date field you need to specify a format in which the date will be show. The format must follow the following table rules:

|  |  |  |
| --- | --- | --- |
| **Format** | **Description** | **Example** |
| ‘d’ | Day of the month, 2 digits with leading zeros | 01 to 31 |
| ‘j’ | Day of the month without leading zeros | 1 to 31 |
| ‘m’ | Numeric representation of a month, with leading zeros | 01 to 12 |
| ‘n’ | Numeric representation of a month, without leading zeros | 1 to 12 |
| ‘Y’ | A full numeric representation of a year, 4 digits | Examples: 1999 or 2003 |
| ‘y’ | A two digit representation of a year | Examples: 99 or 03 |
| ‘g’ | 12-hour format of an hour without leading zeros | 1 to 12 |
| ‘G’ | 24-hour format of an hour without leading zeros | 0 to 23 |
| ‘h’ | 12-hour format of an hour with leading zeros | 01 to 12 |
| ‘H’ | 24-hour format of an hour with leading zeros | 00 to 23 |
| ‘i’ | Minutes, with leading zeros | 00 to 59 |
| ‘s’ | Seconds, with leading zeros | 00 to 59 |

The following is an example of a date field:



## Defining convert functions

A function which converts the value provided by the Reader into an object that will be stored in the Model.

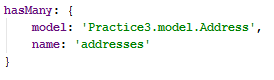
If configured as null, then no conversion will be applied to the raw data property when this Field is read. This will increase performance but you must ensure that the data is of the correct type and does not need converting.

It is passed the following parameters:

* v : Mixed  
  The data value as read by the Reader, if undefined will use the configured defaultValue.
* rec : Ext.data.Model  
  The data object containing the Model as read so far by the Reader. Note that the Model may not be fully populated at this point as the fields are read in the order that they are defined in your fields array

## Defining model associations

Models can have associations with other Models via Ext.data.association.HasOne, belongsTo and hasMany associations.



## Defining model validations

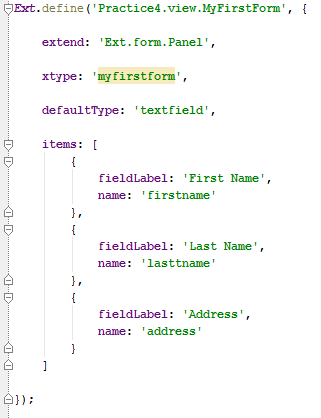
Models have built-in support for validations, which are executed against the validator functions in Ext.data.validations.



# Chapter 5: Working with forms

## Creating a form

FormPanel provides a standard container for forms. It is essentially a standard Ext.panel.Panel which automatically creates a BasicForm for managing any Ext.form.field.Field objects that are added as descendants of the panel. It also includes conveniences for configuring and working with the BasicForm and the collection of Fields.



## Understanding the Anchor layout

This is a layout that enables anchoring of contained elements relative to the container's dimensions. If the container is resized, all anchored items are automatically rerendered according to their anchor rules.

This class is intended to be extended or created via the layout: 'anchor' config, and should generally not need to be created directly via the new keyword.

AnchorLayout does not have any direct config options (other than inherited ones). By default, AnchorLayout will calculate anchor measurements based on the size of the container itself. However, the container using the AnchorLayout can supply an anchoring-specific config property of anchorSize.

If anchorSize is specifed, the layout will use it as a virtual container for the purposes of calculating anchor measurements based on it instead, allowing the container to be sized independently of the anchoring logic if necessary.

## Assigning default configuration values

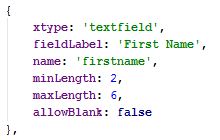
This option is a means of applying default settings to all added items whether added through the items config or via the add or insert methods.

Defaults are applied to both config objects and instantiated components conditionally so as not to override existing properties in the item (see Ext.applyIf).

If the defaults option is specified as a function, then the function will be called using this Container as the scope (this reference) and passing the added item as the first parameter. Any resulting object from that call is then applied to the item as default properties.

## Using TextFields

A basic text field. Can be used as a direct replacement for traditional text inputs, or as the base class for more sophisticated input controls (like Ext.form.field.TextArea and Ext.form.field.ComboBox). Has support for empty-field placeholder values.



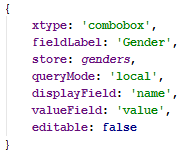
## Using ComboBoxes

A combobox control with support for autocomplete, remote loading, and many other features.

A ComboBox is like a combination of a traditional HTML text <input> field and a <select> field; the user is able to type freely into the field, and/or pick values from a dropdown selection list. The user can input any value by default, even if it does not appear in the selection list; to prevent free-form values and restrict them to items in the list, set forceSelection to true.

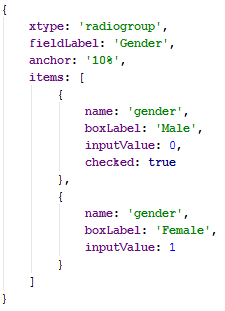
The selection list's options are populated from any Ext.data.Store, including remote stores. The data items in the store are mapped to each option's displayed text and backing value via the valueField and displayField configurations, respectively.

If your store is not remote, i.e. it depends only on local data and is loaded up front, you should be sure to set the queryMode to 'local', as this will improve responsiveness for the user.



## Using Radiobuttons

Single radio field. Similar to checkbox, but automatically handles making sure only one radio is checked at a time within a group of radios with the same name.



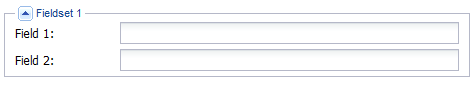
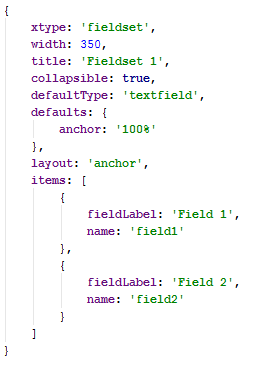
## Using the FieldSet component

A container for grouping sets of fields, rendered as a HTML fieldset element. The title config will be rendered as the fieldset's legend.

While FieldSets commonly contain simple groups of fields, they are general Containers and may therefore contain any type of components in their items, including other nested containers. The default layout for the FieldSet's items is 'anchor', but it can be configured to use any other layout type.

FieldSets may also be collapsed if configured to do so; this can be done in two ways:

1. Set the collapsible config to true; this will result in a collapse button being rendered next to the legend title, or:
2. Set the checkboxToggle config to true; this is similar to using collapsible but renders a checkbox in place of the toggle button. The fieldset will be expanded when the checkbox is checked and collapsed when it is unchecked. The checkbox will also be included in the form submit parameters using the checkboxName as its parameter name.



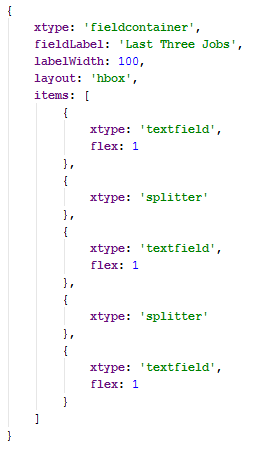
## Using the FieldContainer component

FieldContainer is a derivation of Container that implements the Labelable mixin. This allows it to be configured so that it is rendered with a field label and optional error message around its sub-items. This is useful for arranging a group of fields or other components within a single item in a form, so that it lines up nicely with other fields. A common use is for grouping a set of related fields under a single label in a form.

The container's configured items will be layed out within the field body area according to the configured layout type. The default layout is 'autocontainer'.

Like regular fields, FieldContainer can inherit its decoration configuration from the fieldDefaults of an enclosing FormPanel. In addition, FieldContainer itself can pass fieldDefaults to any fields it may itself contain.

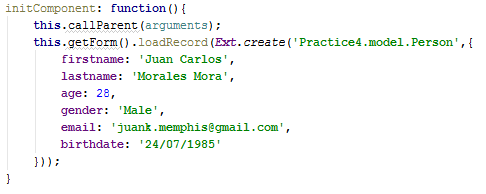
If you are grouping a set of Checkbox or Radio fields in a single labeled container, consider using a Ext.form.CheckboxGroup or Ext.form.RadioGroup instead as they are specialized for handling those types

D:\Dropbox\Documentos\Personal\Software\Trainings\ExtJS 4 Basic to Advanced\Resources\fieldContainer.png

## Linking a form to a model

To link a form to a model you need to do the following:

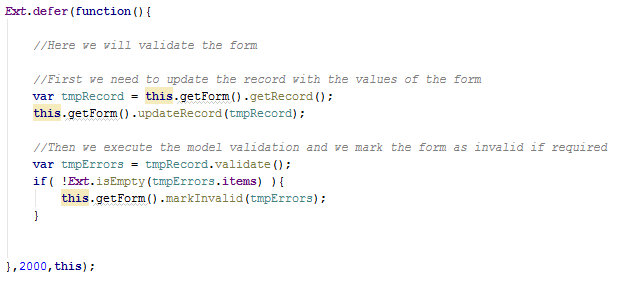
1. First, form field names must be equal to the model field names. So each model field must have its form field.
2. Then you need to load a record in the form using the Ext.form.Basic object.



### Updating and validation the model

To update the model with the values of the form and then validate it, you must do the following:

1. Extract the record from the form
2. Update it with the current form values
3. Validate it

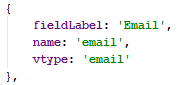


## Working with VTypes

This is a singleton object which contains a set of commonly used field validation functions and provides a mechanism for creating reusable custom field validations. The following field validation functions are provided out of the box:

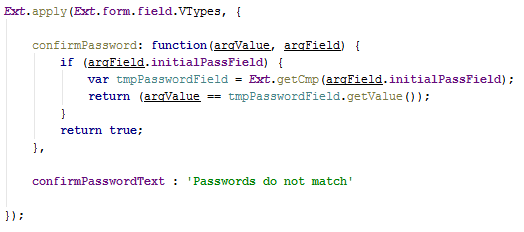
1. alpha
2. alphanum
3. email
4. url

VTypes can be applied to a form Field using the vtype configuration.



## Creating custom vtypes

To create a custom vtype you need to override the Ext.form.field.VTypes class and add you custom vtypes.



# Chapter 6: Working with grids

## Creating a grid

Grids are an excellent way of showing large amounts of tabular data on the client side. Essentially a supercharged <table>, GridPanel makes it easy to fetch, sort and filter large amounts of data.

Grids are composed of two main pieces - a Store full of data and a set of columns to render.



### Configuring columns

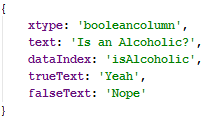
By default, each column is sortable and will toggle between ASC and DESC sorting when you click on its header. Each column header is also reorderable by default, and each gains a drop-down menu with options to hide and show columns.

#### Column types

|  |  |
| --- | --- |
| **Column Type** | **Description** |
| Column | Generic column, usually used to display string values |
| Boolean | Used to output boolean data as strings, e.g. "Yes" or "No" |
| Date | Used to transform Javascript date objects into human-readable date/time values |
| Number | Formats numbers as strings |
| Template | Enables you to use Ext.XTemplate syntax to define the output format of a field |
| RowNumberer | Displays the current row number for a record |
| Action | Enables you to place clickable buttons in a column |

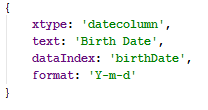
##### Defining a Boolean column

A Column definition class which renders boolean data fields. See the xtype config option of Ext.grid.column.Column for more details.



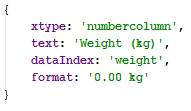
##### Defining a Date column

A Column definition class which renders a passed date according to the default locale, or a configured format.



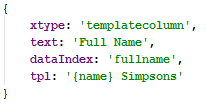
##### Defining a Number column

A Column definition class which renders a numeric data field according to a format string.



##### Defining a Template column

A Column definition class which renders a value by processing a Model's data using a configured XTemplate.



##### Defining a RowNumberer column

A special type of Grid Ext.grid.column.Column that provides automatic row numbering.



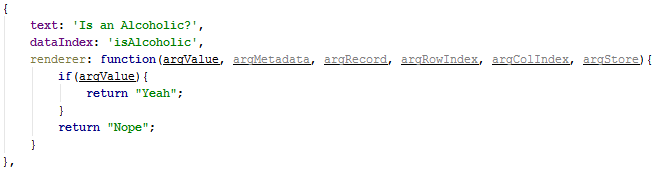
##### Defining an Action column

A Grid header type which renders an icon, or a series of icons in a grid cell, and offers a scoped click handler for each icon.



## Understanding column renderers

A renderer is an 'interceptor' method which can be used to transform data (value, appearance, etc.) before it is rendered.



## Cell editing and row editing

### Cell Editing

The Ext.grid.plugin.CellEditing plugin injects editing at a cell level for a Grid. Only a single cell will be editable at a time. The field that will be used for the editor is defined at the editor. The editor can be a field instance or a field configuration.

If an editor is not specified for a particular column then that cell will not be editable and it will be skipped when activated via the mouse or the keyboard.

The editor may be shared for each column in the grid, or a different one may be specified for each column. An appropriate field type should be chosen to match the data structure that it will be editing.



### Row Editing

The Ext.grid.plugin.RowEditing plugin injects editing at a row level for a Grid. When editing begins, a small floating dialog will be shown for the appropriate row. Each editable column will show a field for editing. There is a button to save or cancel all changes for the edit.

The field that will be used for the editor is defined at the editor. The editor can be a field instance or a field configuration. If an editor is not specified for a particular column then that column won't be editable and the value of the column will be displayed. To provide a custom renderer for non-editable values, use the editRenderer configuration on the column.

The editor may be shared for each column in the grid, or a different one may be specified for each column. An appropriate field type should be chosen to match the data structure that it will be editing.



# Chapter 7: Loading data locally and from a server

## Understanding the Proxy class

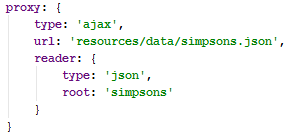
Proxies are used by Stores to handle the loading and saving of Model data. Usually developers will not need to create or interact with proxies directly.

The following are the different types of proxies provided by ExtJS out of the box:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| LocalStorageProxy | Client | Saves its data to localStorage if the browser supports it |
| SessionStorageProxy | Client | Saves its data to sessionStorage if the browsers supports it |
| MemoryProxy | Client | Holds data in memory only, any data is lost when the page is refreshed |
| Ajax | Server | Sends requests to a server on the same domain |
| JsonP | Server | Uses JSON-P to send requests to a server on a different domain |
| Rest | Server | Uses RESTful HTTP methods (GET/PUT/POST/DELETE) to communicate with server |
| Direct | Server | Uses Ext.direct.Manager to send requests |

For more detail on each proxy class please take a look at this link: <http://docs.sencha.com/extjs/4.2.1/#!/api/Ext.data.proxy.Proxy>.

## Defining a proxy to read a local json file



To read a local json file you need to configure an ajax proxy with a json reader and set the url of the local json file. The root property of the reader indicates where in the json the data must be extracted from.

## Defining a proxy to put data in the LocalStorage

The LocalStorageProxy uses the new HTML5 localStorage API to save Model data locally on the client browser. HTML5 localStorage is a key-value store (e.g. cannot save complex objects like JSON), so LocalStorageProxy automatically serializes and deserializes data when saving and retrieving it.

localStorage is extremely useful for saving user-specific information without needing to build server-side infrastructure to support it.

## Defining a proxy to put data in the SessionStorage

Proxy which uses HTML5 session storage as its data storage/retrieval mechanism. If this proxy is used in a browser where session storage is not supported, the constructor will throw an error. A session storage proxy requires a unique ID which is used as a key in which all record data are stored in the session storage object.

It's important to supply this unique ID as it cannot be reliably determined otherwise. If no id is provided but the attached store has a storeId, the storeId will be used. If neither option is presented the proxy will throw an error.

## Defining a proxy to read data from a server

If the server you want to reach is in the same domain of the web app you can use an ajax proxy, otherwise you will need to use a jsonp proxy which allows you to read data from a server located in a different domain.

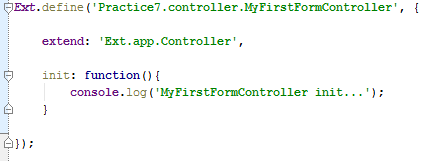
# Chapter 8: Defining Controllers

## Creating a Controller

Controllers are the glue that binds an application together. All they really do is listen for events (usually from views) and take some action

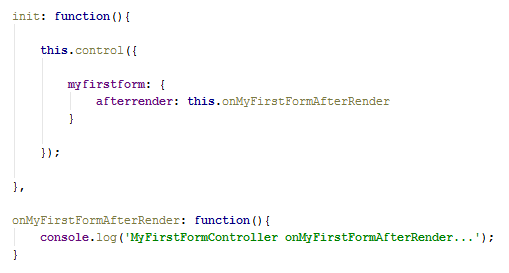
The init function is a special method that is called when your application boots. It is called before the Application's launch function is executed so gives a hook point to run any code before your Viewport is created.

The init function is a great place to set up how your controller interacts with the view, and is usually used in conjunction with another Controller function - control. The control function makes it easy to listen to events on your view classes and take some action with a handler function.



## Defining controllers listeners

To add listeners to a controller you need to call the control function and pass as parameter an object in the format of key-value. The key will be the query of the component we want to hear and the value will be another object with the events we want to hear.



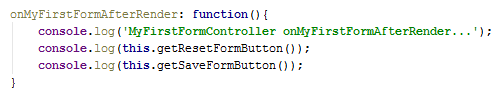
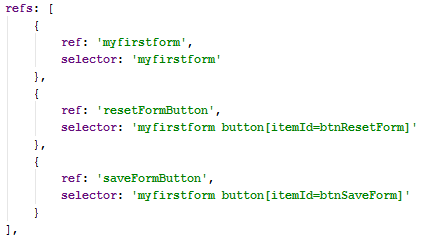
## Defining controllers refs

One of the most useful parts of Controllers is the ref system. These use the Ext.ComponentQuery to make it really easy to get references to Views on your page.

Every ref must have

* selector: the query that we will use to find a specific component
* ref: the name that we want to set to this reference

ExtJS will generate a getter function for each reference based on the name of our ref, which was capitalized and prepended with get.



# Chapter 9: Working with templates

## Understanding the XTemplates class

The Ext.XTemplate class generates markup from abstract data structures. XTemplate syntax supports the following:

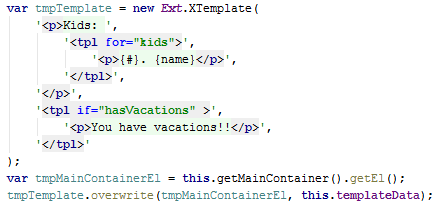
* Automatic looping through arrays and records
* Conditional processing with if and switch constructs
* Executing inline Javascript code
* Custom template functions
* Basic math function support

Templates can be bound to an Ext.panel.Panel or to an Ext.view.View (xtype: dataview). The primary difference between these view classes is the following:

* Panels automatically update whenever their setData() method is explicitly invoked.
* Ext.view.View instances are bound to an Ext.data.Store and automatically refresh their contents whenever data in the Store is changed.

## Implementing conditional processing

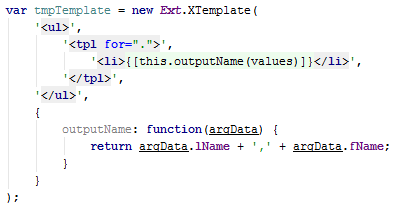
Use <tpl if> to implement branching as illustrated below:



## Binding custom javascript methods to XTemplates

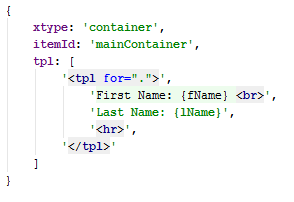
You can bind custom JavaScript methods to your template in order to implement functionality not directly supported by <tpl> constructs.

Custom methods are attached to the template as the final argument to the XTemplate constructor as illustrated by the following code snippet:

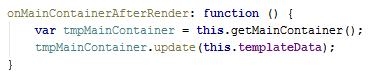


## Binding a template to a component

You can bind a template directly to a container via its tpl config property as illustrated below:



Panels that are bound to a template will automatically refresh their markup whenever the update() method is called as illustrated by the following:



## Explaining the DataView component

The View uses an Ext.XTemplate as its internal templating mechanism, and is bound to an Ext.data.Store so that as the data in the store changes the view is automatically updated to reflect the changes. The view also provides built-in behavior for many common events that can occur for its contained items including click, doubleclick, mouseover, mouseout, etc. as well as a built-in selection model. In order to use these features, an itemSelector config must be provided for the View to determine what nodes it will be working with.



# Chapter 10: Creating custom components

## Understanding components life cycle

The main 3 functions that you need to override when extending an extjs component are:

* initComponent:   
   The initComponent template method is an important initialization step for a Component. It is intended to be implemented by each subclass of Ext.Component to provide any needed constructor logic.   
   The initComponent method of the class being created is called first, with each initComponent method up the hierarchy to Ext.Component being called thereafter. This makes it easy to implement and, if needed, override the constructor logic of the Component at any step in the hierarchy.   
   The initComponent method must contain a call to callParent in order to ensure that the parent class' initComponent method is also called.   
   All config options passed to the constructor are applied to this before initComponent is called, so you can simply access them with this.someOption.
* afterRender:   
   Here the component was already rendered. Is a good place to add additional behavior after rendering is complete. At this stage the Component’s Element will have been styled according to the configuration, will have had any configured CSS class names added, and will be in the configured visibility and the configured enable state.
* destroy:   
   Executed when the component is about to be destroyed.

# Chapter 11: Creating a production build

## Using Sencha CMD to generate a production build

To generate a production or testing build is really easy, you just need to run the following commands accordingly:



Generates a testing build, not minified or obfuscated



Generates a production build, minified and obfuscated.

# Chapter 12: Best Practices

Please refer to the power point presentation for detail on the best practices for extjs.