DZone Refcardz

CONTENTS INCLUDE:

Sencha Touch

Open-Source HTML5 Mobile Web Development Made Easy

By: Stan Bershadskiy

WHAT IS SENCHA TOUCH?

Sencha Touch is an open source mobile HTML5 application framework. It allows developers to create rich native-like applications targeting Android, iOS, Blackberry, and KindleFire devices. These applications can be displayed either in the device's browser or wrapped as a hybrid application using Sencha's Packager or PhoneGap (Cordova).

Sencha Touch is compatible with all WebKit based browsers. Even though this includes desktop browsers, such as Apple Safari and Google Chrome, Sencha Touch is not designed for these browsers. For desktop Rich Internet Applications, there exists Ext JS, which is also by Sencha.

Sencha Touch or jQuery Mobile?

These two frameworks are often put alongside each other in comparisons, although they are different and serve different purposes. jQuery Mobile does a great job of creating mobile friendly web sites. Sencha Touch specializes in mobile web applications and supports the complete application lifecycle internally. Sencha Touch can be downloaded for free from Sencha at www.sencha.com/products/touch/

What's inside Sencha Touch?

Sencha Touch is built on several core subsystems:

- **Class system**
- **Rich Component and Layout system**
- **Event system**
- **Data Package**
- **MVC Architecture**
- **Theming**

These systems and their related topics will be covered in this Refcard. This Refcard will serve as a companion to the Sencha Touch documentation by providing quick reference and useful examples. All the material applies to Sencha Touch 2+, there may be differences with each minor version, but all the depicted examples should be functional.

THE CLASS SYSTEM

The cornerstone of Sencha Touch is its class system, which was originally introduced in ExtJS 4. The class system provides support for inheritance, dependency injection, configuration options, mixins, and more.
Here is an example of a couple of classes leveraging most of the Classt system's features:

```
Ext.define('Car', {
    config      : {
        make     : null,
        model    : null,
        price    : null,
        topSpeed : null
    },
    constructor : function (config) {
        this.initConfig(config);
    },
    isFast      : function () {
        return this.getTopSpeed() > 60;
    },
    isExpensive : function () {
        return this.getPrice() > 50000;
    }
});

Ext.define('ExoticCar', {
    extend : 'Car',
    config : {
                     ------Snippet cont'd on next column-------->
```

```
        seriesNumber : null
    },
    isFast : function () {
        return this.getTopSpeed() > 150;
    },
    isRare : function () {
        return this.getSeriesNumber() < 5;
    }
});
```

Here we define a Car class and ExoticCar subclass. As you can see, the Car has some basic properties defined in the config object. There are also a couple of functions implemented to do some basic calculation. An ExoticCar is a type of a Car, thus it extends it and adds its own specific implementation.

We can see this in action by instantiating the classes. To instantiate a class, we use Ext.create instead of the new keyword. Using the former leverages Sencha Touch's dynamic dependency loader and allows for using aliases, as well as easily applying the config object.

```
var toyotaCamry = Ext.create('Car', {
    make     : 'Toyota',
    model    : 'Camry',
    price    : 20000,
    topSpeed : 40
});
toyotaCamry.isFast();   // return false
toyotaCamry.getPrice(); // returns 20000

var ferrariF458 = Ext.create('ExoticCar', {
    make         : 'Ferrari',
    model        : 'F458 Italia',
    price        : 200000,
    topSpeed     : 208,
    seriesNumber : 1
});
ferrariF458.isFast();      // returns true
ferrariF458.getMake();     // returns "Ferrari"
ferrariF458.isExpensive(); // returns true
```

You may notice the use of some functions that we have not explicitly defined, for example getMake(). The Sencha Touch class system does us a favor and generates 4 types of convenience functions:

AnswerHub
The Enterprise Q&A Platform
Work Smarter, Not Harder
Connect your entire organization with fast, accurate answers.
Learn More

- **Getter – return current value; getName()**
- **Setter – set new value; setName()**
- **Applier – called by setter, execute logic when property changes; applyName()**
- **Updater – called by setter, execute logic when the value for a property changes; updateName()**

Hot Tip

Please note that the applier and updater functions require implementation. The applier function requires a return of the value that will be set to the property. The updater gets fired after the applier and does not handle a return, rather, it just informs of a value change.

Let's assume we want to round the price of the Car to the nearest 1000. To do so we would implement the applyPrice function in the Car class:

```
applyPrice : function (newPrice, oldPrice) {
        return Math.round(newPrice / 1000) * 1000;
    }
```

Usage:

```
toyotaCamry.setPrice(21995);
toyotaCamry.getPrice(); // returns 22000
```

THE COMPONENT SYSTEM

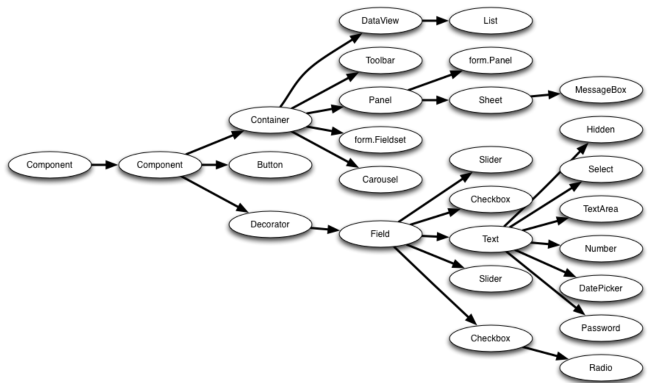
The Sencha Touch Component system is the foundation of all the visual elements in the framework. The Component system makes full use of the Class system's dependency injection and hierarchical support. Every Sencha Touch view class is a subclass of Ext.Component, and thus follows its lifecycle. Components are generally defined by an xtype. An xtype is a shortcut for the full component Class name allowing for lazy instantiation. Using xtypes to instantiate components allows the creation of the object to be deferred to when it's actually needed, thus saving resources.

There may be times when you want to display multiple components, either all at once or individually. The Ext.Container class exists for these situations. Containers provide functionality for adding child Components, removing Components and specifying a Layout that determines how these components are displayed.

Layouts

Type	Description
Box	Implements the flexible box layout, either 'hbox' or 'vbox'. Aligns components on a given axis, component widths defined by flex values associated with the component.
Card	Displays only one component at a time, with each component centered and sized to fill the container.
Fit	Makes the child component fit the full size of its container. Note: Only the first child is visible, use Card Layout for multiple children.

Common Components



Component	Purpose	Useful Properties & Functions
Component Ext.Component xtype: component	Basic Visual Class	xtype cls html tpl data show() / hide() / destroy()
Container Ext.Container xtype: container	Manages Component rendering and arrangement	layout items scrollable add(component) / remove(component)
Panel Ext.Panel xtype: panel	Container designed to float as overlays	showBy(component)
Toolbar Ext.Toolbar xtype: panel	Docked containers that generally hold Buttons and a Title	title left / right ui
Button Ext.Button xtype: button	Basic tappable component	text ui iconCls iconAlign
Tab Panel Ext.tab.Panel xtype: tabpanel	Container that allows user to easily switch between components using a Toolbar	tabBarPosition

MessageBox
Ext.Msg
Ext.MessageBox
	Modal Container for displaying alerts and notifications. Ext.Msg is a singleton that allows for creation of predefined MessageBox instances.	Ext.Msg.alert(title, msg, callback)
		Ext.Msg.confirm(title, msg, callback)
		Ext.Msg.prompt(title, msg, callback)

Carousel Ext.Carousel xtype: carousel	Container that allows user to switch active components by swiping	ui direction indicator next() / previous()
List Ext.dataview.List xtype: list	Container of dynamically rendered components from a Data Store	store itemTpl itemCls onItemDisclosure grouped indexBar

Form Panel
Ext.form.Panel
xtype: formpanel
	Panel that contains a set of form fields to be displayed	getValues() / setValues(data)
		submit(options, params, headers)

Form Fields Ext.field.Field xtype: field	Base class for all data input form fields	name label / labelCls / labelAlign inputCls required

THE EVENT SYSTEM

A crucial part of the Component and Application life cycle in general are the events that are fired at different stages. These events can be triggered from some interaction with a component or be fired after some arbitrary business logic is executed. Most importantly, for each event we can configure simple listeners to handle these events.

We react to events by attaching event listeners on a given component through the listeners property or the on() function where we pass in an object where the event name maps to a callback. When we define the listeners, we can also attach the context that the callback gets executed in via the scope property.

```
myButton.on({
    tap   : function () {
        console.log('button tap');
    },
    scope : this
});
```

Common Events Fired by Components

- **painted – fires when element becomes rendered on the screen**
- **show / hide – fires when view component, generally a modal, is shown or hidden using show() and hide()**
- **updatedata – fires when data of component is updated**
-

Common Events Fired by Containers

- **activate / deactivate – fires when an item in the container is activated or deactivated**
- **activeitemchange – (card layout only) fires when the actively visible item is changed by setActiveItem()**
- **add / remove – fires when components are added or removed from container**

Sencha Touch allows us to also manage events that are fired by the DOM elements that these components render. These DOM elements are of type Ext.dom.Element. The elements fire events for native browser gesture interactions.

Here is an example of attaching event listeners for both component and element events. You can also see some simple event handler implementation as well as firing of custom events.

```
Ext.define('MyApp.view.tooltip.Tooltip', {
    extend       : 'Ext.Panel',
    xtype        : 'tooltip',
    config      : {
        /* ... */
    },
    initialize   : function () {
        var me = this;

        me.element.on({
            tap        : me.onTap,
            touchstart : me.onTouchStart,
            touchend   : me.onTouchEnd,
            scope      : me
        });

        me.on({
            show  : me.onShow,
            hide  : me.onHide,
            scope : me
        });

        me.callParent();
    },
    onTap        : function (evtObj) {
        this.fireEvent('close');
    },
    onTouchStart : function (evtObj) {
        var closeButton = evtObj.getTarget(".close-button");
        if (closeButton) {
            Ext.fly(closeButton).addCls('pressed');
        }
    },
    onTouchEnd   : function (evtObj) {
        var closeButton = evtObj.getTarget(".close-button");
        closeButton && Ext.fly(closeButton).removeCls('pressed');
    },
    onShow       : function() {
        /* ... */
    },
    onHide       : function () {
        this.destroy();
    }
});
```

Common Events Fired by Elements

- **tap - fires for every tap**
- **doubletap - fires for a double tap**
- **taphold / longpress - fires for when you hold a tap for > 1 second**
- **touchstart / touchend - fires for when the touch starts and finishes**
- **drag - continuously fires for dragging**
- **dragstart / dragend - fires when dragging begins and finishes**
- **pinch / rotate - continuously fires for pinch and rotate gestures**
- **swipe - fires when there is a swipe; the event property holds the swipe direction**

SELECTORS

One of the most useful functionalities of Sencha Touch and ExtJS as well is the support for finding the instance of a component or DOM element. Similar to jQuery's $ selector, Sencha Touch provides the child(), up(), down() and query() functions to each Container class. These functions take in a ComponentQuery Selector and return either the first component (child(), up(), down()) or an array of all matching components (query()). You can also query for components on a global scope by using the Ext. ComponentQuery singleton, which has a query() method.

Selector Type	Purpose	Example

xtype	Select component based on its xtype	Ext.ComponentQuery.query('.formpanel');
		
		 myContainer.down('.button');

itemId / id	Select component based on its itemId or id property	Ext.ComponentQuery.query('#myFirstPanel');
		
		 myContainer.down('#submit');

Attribute	Select a component based on its xtype and some property value that is set on it	myFormPanel.down("field[name=userName]);
Nested Selector	Select a child of a matched selector	Ext.ComponentQuery.query('panel > toolbar');
Multiple Selector	Select children that match multiple selectors	Ext.ComponentQuery.query('formpanel, carousel');

Hot Tip

We can also query on DOM Elements. The Ext.dom.Element class provides query() and down() functions. Here however we use CSS selectors. More information on CSS Selectors can be found here: http://www.w3.org/TR/selectors/

DATA PACKAGE

A crucial requirement for many mobile applications is to consume data from external sources as well as maintain internal application specific data. For these situations Sencha Touch provides the Data Package, a set of classes responsible for loading, storing, manipulating and saving data.

Three types of classes generally leverage the data package:

- **Model – an entity that represents a single record, defining its fields, validation, associations and conversion parameters.**
- **Store – collection of model instances with support for sorting, filtering, and grouping**
- **Proxy – medium that interacts with the remote or local data source and performs the necessary reading and writing operations.**

Models

Models are data objects that are managed in your application. Models are defined by a set of data fields. Each field is defined by its name, type, and optional defaultValue.

Supported field types are:

- **string**
- **int**
- **float**
- **boolean**
- **date**

A convert function can also be implemented on the field to perform processing on the field's data.

Here is a sample model:

```
Ext.define('MyApp.model.Car', {
    extend : 'Ext.data.Model',
    config : {
        fields : [
            {
                name : 'make',
                type : 'string'
            },
            {
                name : 'model',
                type : 'string'
            },
            {
                name : 'topSpeed',
                type : 'int'
            },
            {
                name : 'price',
                type : 'float'
            },
            {
                name : 'options',
                type : 'auto'
            }
        ]
    }
});
```

Data Proxies

There are four proxies provided by Sencha Touch that can be used in a Store or Model.

- **LocalStorage – saves data to localStorage**
- **Memory – saves data in memory, extremely volatile**
- **Ajax – communicates with a RESTful resource on the same domain**
- **JsonP – communicates with a service on an external domain using JSON-P**

Each proxy type implements four basic operations create, read, update, and destroy.

When interacting with Server Proxies, there is a need for serializing the data sent and retrieved from the server. For this Sencha Touch provides the JsonReader, JsonWriter, XmlReader and XmlWriter classes. These reader and writer types are specified on the proxy definition.

Example Store with Proxy

```
Ext.define('MyApp.store.Cars', {
    extend   : 'Ext.data.Store',
    requires : [
        'Ext.data.reader.Json',
        'Ext.data.writer.Json'
    ],
    config   : {
        model : 'MyApp.model.Car',
        proxy : {
            type   : 'ajax',
            url    : '/cars',
            reader : {
                type         : 'json',
                rootProperty : 'cars'
            }
        }
    }
});
```

Hot Tip

Proxies can be attached to both Models and Stores.

To access a store from anywhere in the application you can use the Ext.getStore() function and pass in either a defined storeId or the store's classname, excluding namespace.

Example of loading data into a store:

```
Ext.getStore("Cars").load();
```

XTEMPLATE

Template based components have been the norm in many web applications thanks to frameworks like Mustache, Handlebars, Closure, and more. Sencha Touch provides first class functionality for templating through the Ext.XTemplate class.

The Ext.XTemplate class supports not only simple data identifiers but also:

- **Array traversal**
- **Conditional processing**
- **Basic math functionality**
- **Execute arbitrary code**
- **Execute custom template functions**

When defining an XTemplate, the tpl tag is reserved for template commands.

Array Traversal

Looping through an array is done with the tpl for keyword with the value being either "." or a property path.

Hot Tip

While iterating the array, there are two special variables {#} and {.}. {#} represents the current index of the array + 1. {.} represents the value of the element at the current index.

Example assuming the data is an array of Car objects:

```
<tpl for=".">
    <div>{make} {model}</div>
</tpl>
```

Example assuming the data is an object with the a cars property that holds arrays grouped by make:

```
<tpl for="cars.toyota">
    <div>{#} {.}</div>
</tpl>
```

Conditional Processing

XTemplate supports 4 basic comparison operations: if, else-if, else, and switch.

Examples:

```
<tpl for="cars">
    <div>{make} {model} is
    <tpl if="topSpeed &gt; 150">
        <span>ridiculously fast</span>
    <tpl elseif="topSpeed &gt; 80">
        <span>somewhat fast</span>
    <tpl else>
        <span>ridiculously slow</span>
    </tpl>
    </div>
</tpl>

<div> {make} is
<tpl switch="{make}">
    <tpl case="BMW" case="Mercedes-Benz" case="Audi">
        <span>luxurious</span>
    <tpl default>
        <span>economical</span>
</tpl>
```

Math Support

The following math operators are supported: + - * /

Example:

```
<tpl for="cars">
    <div>Top Speed in km/h is {topSpeed * 1.6}</div>
</tpl>
```

Executing Arbitrary Code

It is possible to execute JavaScript code while in the scope of the template by using [] inside an expression.

While executing code inside a template expression there are several special values available:

- **out – output array of the template**
- **values – values in the current scope**
- **parent – values of parent data object**
- **xindex – if in looping template, then the 1-based index**
- **xcount – if in looping template, then the total length of array**

Example:

```
<tpl for="cars">
    <div>{[values.make.toUpperCase() + " –  " + values.model.
toLowerCase()]}</div>
</tpl>
```

Custom Template Functions

When defining our Ext.XTemplate instance, we can implement functions that will be executed by the template.

Example:

```
var tpl = new Ext.XTemplate(
    '<tpl for="cars">',
        '<div> {make} is',
        '<tpl if="this.isFast(make)">',
            '<span>fast!</span>',
        '<tpl else>',
            '<span>slow</span>',
        '</tpl>',
    '</tpl>',
    {
        isFast : function (make) {
            return (make === "Ferrari");
        }
    }
);
```

MVC

Previously we introduced Views and Models. Now, it's time to connect them together to form a full-fledged application by adding in a Controller concept.

Controller

Sencha Touch implements the Controller through the Ext.app.Controller class. The Controller is responsible for responding to events that your components may fire. There are two core configuration properties of the Controller: ref and control.

- **refs – execute ComponentQuery to locate first component instance to match selector and generate getter function**
- **control – listen to events from ComponentQuery selectors or refs and attach listeners**

Controller Example:

```
Ext.define('MyApp.controller.Main', {
    extend              : 'Ext.app.Controller',
    config              : {
        refs    : {
            loginPanel  : 'loginpanel',
            loginButton : 'loginpanel button[text="login"]'
        },
        control : {
            loginpanel  : {
                validate : 'onLoginPanelValidate'
            },
            loginButton : {
                tap : 'onLoginButtonTap'
            }
        }
    },
    onLoginPanelValidate : function () {
        this.getLoginPanel().validate();
    },
    onLoginButtonTap     : function () {
        this.getLoginPanel().submit();
    }
});
```

Application

Now that we have Models, Views, and Controllers defined we need a way to connect them together so that they can coexist and work with each other. For this we have the Ext.app.Application class that connects Models, Controllers, Profiles, Stores and Views together by instantiating them through the dependency loader.

All Sencha Touch applications begin with a call to the Ext.application function that creates an Application instance.

Example Application instance:

```
Ext.application({
    name        : 'MyApp',
    models      : [
        'Car',
        'User'
    ],
    stores      : [
        'Cars',
        'Users'
    ],
    views       : [
        'LoginPanel',
        'CarDetailView'
    ],
    controllers : [
        'Main'
    ],
    launch      : function () {
        Ext.fly('appLoadingIndicator').destroy();
    }
});
```

THEMING

An important aspect of any application is styling and theming. Sencha Touch applications are styled using SASS and compiled with Compass.

Hot Tip

Compass requires Ruby to be installed on your computer and can be installed using rubygems. More information can be found here: http://compass-style.org/ More information on SASS can be found at: http://sass-lang.com/

Common Global SASS Variables

Variable	Description
$base-color	Primary color that most elements reference
$base-gradient	Primary gradient color most elements reference
$font-family	Font family used throughout the theme
$page-bg-color	Background color of fullscreen components

Common Global SASS Mixins

Mixin	Description
pictos-iconmask($name)	Include a base64-encoded icon to be used with Tab Buttons. Name must match icon's file name in resources/themes/images/default/pictos without extension @include pictos-iconmask('wifi2');
bevel-text($type)	Add text shadow or highlight. $type is either shadow or highlight.
stretch()	Stretch an element to parent's bounds
ellipsis()	Apply element text-overflow to use ellipsis

PACKAGING / DEPLOYING

Since Sencha Touch is a full-fledged application framework it is only fitting that there exists a tool supporting the application's entire life cycle.

Sencha Cmd is a command line utility that provides automated tasks from generating a new application to generating components of an existing application to preparing an application for production deployment.

Sencha Cmd contains the following major features:

- **Code generation**
- **Optimized Sencha specific JavaScript minifier and obfuscator**
- **Native Packager to wrap mobile web application as a native hybrid application**
- **Workspace management allows sharing code between applications**

Hot Tip

You may notice that when we specify the models, stores, views, and controllers we do not include the necessary namespaces. You can also specify models, stores and views in Controllers. In the Ext.app.Application and Ext.app.Controller class, Sencha Touch's loader looks up the dependency based on the context provided in the configuration.

Example Usage

Generate New Application

```
Sencha generate app MyApp /path/to/MyApp
```

Generate Models, Controllers, and Views

```
cd /path/to/MyApp
sencha generate model Car make:string,model:string,price:float
sencha generate controller Main
sencha generate view CarDetailView
```

Build Application

```
sencha app build [–d ‘testing’|’production’|’package’|’native’]
```

COMMON PITFALLS

Hardcoded ids and Ext.getCmp

Sencha Touch generates a unique random id and registers it with the ComponentManager. It is possible to force a static id for your component and retrieve it using the Ext.getCmp(id) function. However, as your project and number of developers grow, the chance that the static ID that you thought of stays unique or diminishes. Thus, it is recommended to use either itemId, which assigns a pseudo-id and is scoped locally to the Container, or to use ComponentQuery.

Controller Refs with multiple instances

Generally, it is a good practice to make your components generic enough to be reusable or extensible throughout your application. Often, when defining a Controller ref, it is forgotten that it only returns the first instance that matches the ComponentQuery selector. Thus, it is recommended to use a ComponentQuery inside your event listener to determine the proper instance of the desired Component you need.

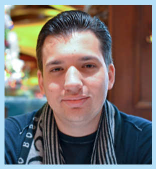
Over-nesting Containers

Sencha Touch Containers do an excellent job in maintaining their child components and making sure they are displayed properly. To accomplish this task, there is a lot of calculation and manipulation of the DOM involved in the layout managers of these containers. It is easy to begin carelessly nesting these containers inside of each other. When the containers are nested it becomes much more processor intensive to calculate how to display the components and their underlying DOM elements properly.

Relying on Built-in Components

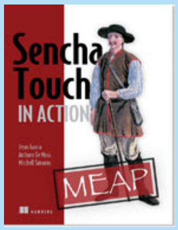
Many of Sencha Touch's components are designed to be widgets that are meant to be items of a Container and arranged by the layout manager. While this is acceptable in certain simple situations, it often results in unnecessary utilization of the DOM. Sencha Touch allows us to create custom views by leveraging XTemplate and controlling the markup of our component to be as minimal as necessary.

ABOUT THE AUTHORS



Stan would like to thank Jay Garcia (CTO, Modus Create) and Grgur Grisogono (Principal, Modus Create) for their support and contribution. Originally a Java developer for a large enterprise he desired a change and joined Modus Create. Stan has been working with Sencha products since 2008, originally with their Google Web Toolkit (GWT) offerings. Lately he has been deeply involved with Sencha Touch and has developed apps that are live on the App Stores. He is located in New York City and can usually be found either presenting or spectating at the NYC JS Meetup sponsored by Modus Create (http://moduscreate.com).

RECOMMENDED BOOK



Sencha Touch, a mobile framework for HTML 5 and JavaScript, enables developers to build truly cross-platform mobile apps or to extend existing enterprise applications to mobile clients. With numerous features that mimic the native capabilities of mobile platforms and a MVC architecture that feels right at home for application developers, Sencha Touch is the most efficient way available to build sophisticated, high-performance rich mobile web applications.

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome

refcardz@dzone.com

Sponsorship Opportunities

sales@dzone.com

ISBN-13: 978-1-936502-78-3
ISBN-10: 1-936502-78-X
50795
9 781936 502783
$7.95

Version 1.0