Sencha Touch IN ACTION

Jesus Garcia
Anthony De Moss

MEAP

MANNING

MEAP Edition
Manning Early Access Program
Sencha Touch in Action version 1

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

1

Introduction to Sencha Touch

This chapter covers

- The problems Sencha Touch aims to solve
- The Sencha Touch UI palette
- Thinking like a mobile web developer

If you're like me, you are on the hook to build a mobile application. Perhaps you've been tasked in a project or have a great idea and want to make it become a reality. Either way, to build this application, you're going to at least have to learn Objective C for iOS or Java for Android. It should be no surprise that if you're to support both types of devices, you'll have to learn and master both languages, unless you choose a third party native framework to bridge the gap between the devices.

Chances are you have experience in HTML, CSS and JavaScript and want to leverage what you know to build your mobile application. This is part of what makes Sencha Touch a good choice for folks like you and I, since it offers a wide range of UI widgets to choose from, as well as robust data, layout and component models.

In this chapter, we'll begin our journey into the world of Sencha Touch, where we'll discuss what Sencha Touch is and what problems it aims at solving. We'll then look at the widgets that the framework provides. Lastly, we'll discuss some of the ways you should think about developing your mobile application, in hopes to avoid future performance issues.

What we'll find along the way is that developing mobile applications with this framework is not as difficult as with other technologies, such as Objective C or Java.

1.1 What is Sencha Touch?

Sencha Touch was born out of many of the ideas and culture from the venerable Ext JS framework and is the first mobile HTML5 JavaScript framework. Sencha Touch aims at solving

the problem of having mobile web applications that mimic natively compiled applications while making full use of HTML5 and CSS3. Sencha Touch runs on iOS (iPhone, iPad) devices, as well as Android phones and tablets.

read about html5

> HTML5 is changing the way we develop web applications. While it's not completely necessary to know everything about HTML5 to use Sencha Touch, it is a good idea to get the basics down. A great site to learn about HTML5 is http://www.html5rocks.com/.

An excellent example of a Sencha Touch application is Checkout, by Steffen Hiller, illustrated below running on an iPad.

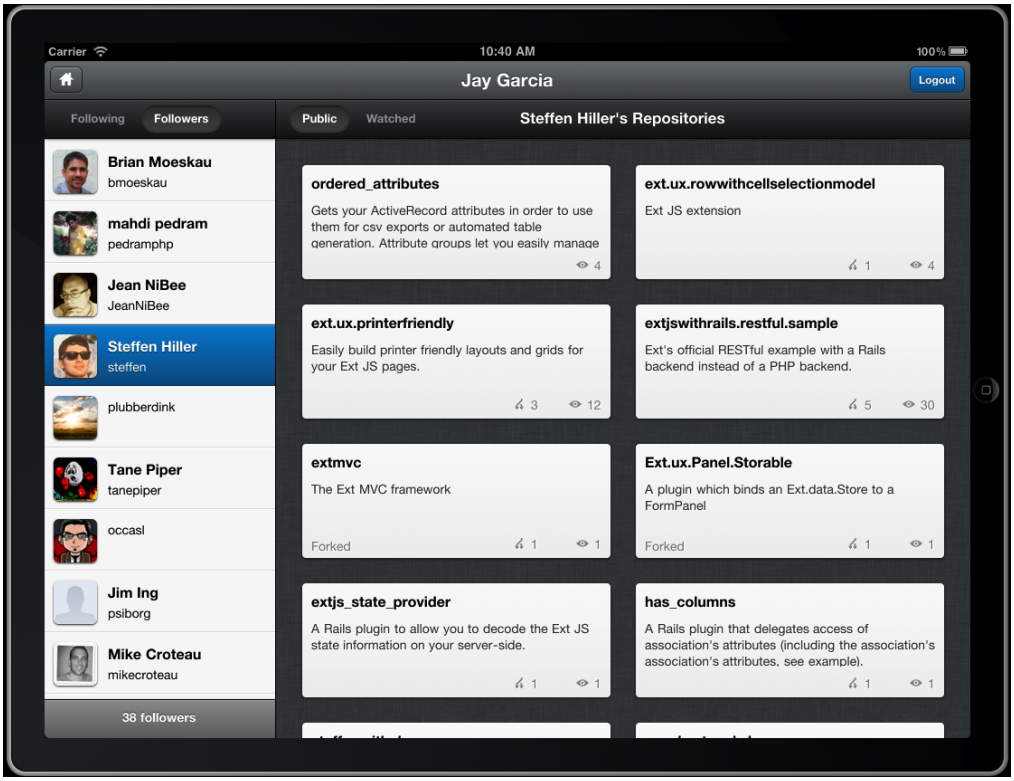

Figure 1.1 A full-screen Sencha Touch application, known as Checkout, which allows you to keep tabs on your Github account and followers. You can learn more about it via http://checkout.github.com.

Much like Ext JS, Sencha Touch brings the native application feel by means of a clever blend of HTML5, CSS3 and JavaScript, all optimized for the best mobile experience possible given the constraints of mobile devices today.

Also like its big brother, Sencha Touch is designed to be extensible and modifiable out of the proverbial box.

1.1.1 What Sencha Touch is not

While Sencha Touch works on desktop WebKit browsers, like Safari and Chrome (to a limited capacity), it is not designed for desktop Rich Internet Applications. Upon its release, lots of developers balked at the idea of this framework not functioning in Firefox or Internet Explorer.

The fact is that Sencha Touch is aimed at the development of mobile applications only. This means that if you've only developed applications with Firefox, IE and their respective debugging toolkits, you're going to have to leave your comfort zone.

"Sencha Touch" !== "Ext JS"

If you're a veteran Ext JS developer, you will feel right at home when learning Sencha Touch. It is important to know that there are some significant differences between the two libraries. Throughout this book, we'll try to point out some of the differences, but cannot cover every possible point. If you have doubts, always check with the API documentation.

1.1.2 Lots of wiring under the hood

To make user of mobile interactions, Sencha Touch comes with a gesture library, allowing you to easily hook into gesture-based events, such as tap, pinch and swipe. One way Sencha Touch to comes close to the feel of native applications is by means of a custom physics-based Scroller class, which uses hardware-accelerated CSS3 transitions and includes key variables like slide friction and spring effects.

1.1.3 Hardware compatibility

Many mobile touch-screen smart devices are entering the marketplace today and is helping drive the increase in demand for mobile applications. While Sencha Touch aims at 100% compatibility across all mobile devices, the best user experience is on iOS and high-powered android devices.

Why the difference in user experience?

The main reason for the difference in user experience between iOS and Android has to do with the physical computing power of each device and how the each individual device manufacturer compiles mobile WebKit for their device. Apple devices include a GPU, and compiles mobile Safari with GPU acceleration enabled. Most android devices do not have dedicated GPUs. And for the ones who do, manufacturers typically do not compile mobile WebKit to enable GPU acceleration.

Sencha Touch applications do such a great job at mimicking how native applications look and feel that it's often easy to get lost in the fact that you're using a web-based application. This especially holds true when the mobile WebKit toolbars are hidden from view.

1.1.4 Full-screen goodness

Figure 1.2 illustrates how a Sencha Touch application looks when accessing the application via mobile Safari (left), versus accessing the application via a shortcut on your home screen.

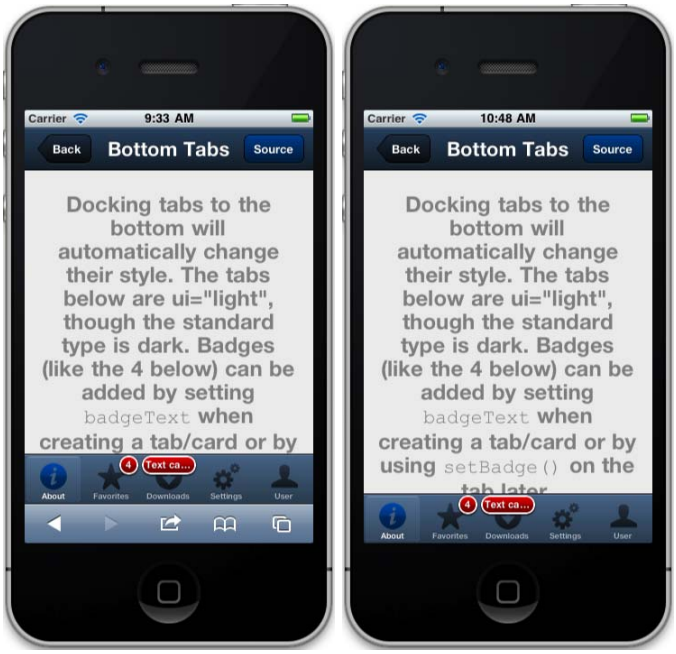

Figure 1.2 Looking at the Sench Touch kitchen sink example via Mobile Safari directly (left), or a shortcut on your home screen (right).

After looking at figure 1.2, it should be clear that a full-screen view of a Sencha Touch application looks very close to a native application. Also having your app in full screen means that much needed screen real estate is usable in your applications.

Sencha Touch offers a lot when it comes to UI widgets, but it's certainly just the surface of this framework. If you're like me, you probably want to just skip ahead and dive into code. But before we get our hands dirty, I think we should browse through the library and discuss some of its features.

1.2 Browsing the Sencha Touch UI

If you have glanced at the Sencha Touch API documentation, the sheer number of classes in the library might have overwhelmed you. To make sense of it all, we must understand that these classes can be broken down into a few major groups.

The table below describes the major groups that Sencha Touch is broken down in.

Table 1.1 Describing the various sections of purpose in the Sencha Touch framework.

Group	Purpose
Platform	The shared base of Sencha Touch and Ext JS v4 and is the bulk of the code for Sencha Touch.
Layout	A set of managers for visually organizing widgets on screen.
Utilities	Utilities is a group of useful odds and ends for the framework.
Data	Data is the information backbone for Sencha Touch and includes the means for retrieving, reading and storing data.
Style	Sencha Touch's theme is automatically generated via Sass (Syntactically Awesome Style Sheets).
MVC	Sencha Touch comes with an MVC framework for your application.
UI Widgets	A collection of visual components that your users will interact with.

The base library for Sencha Touch is known as Sencha Platform. Sencha Platform is shared between Ext JS 4.0 and Sencha touch, as it contains a lot of the common code between the two frameworks.

The Layout portion of Sencha touch is implemented by some of the UI widgets and is the code responsible for visually organizing items on the screen. The layouts are responsible for implementing transitional animations if configured to do so.

The Utilities section of the framework is a collection of useful bits of functionality that are often implemented by the framework and can be implemented by you. For instance, the List widget implements XTemplate to paint HTML fragments on screen. However, the XTemplate is open for you to use to do the same in your own custom widget.

The Data package is a group of classes that gives Sencha Touch the ability to fetch and read data from a myriad of sources, including mobile WebKit's HTML5 Session, Local and Database Storage methods. Sencha Touch can read data in a variety of formats including XML, Array, JSON, and Tree (nested).

The style area of the framework is not something that you typically deal with on a day-to-day basis, but is something that is worth mentioning. From the very beginning, Sencha Touch has implemented Sass to allow easy style changes to the UI. This means if you want to change your entire color scheme, you can do so with relative ease if you know Sass.

LEARN MORE ABOUT SASS

> Sass has taken the world of style sheet management by storm and has arguably revolutionized how people style their web pages and apps. To learn more about this utility, check out http://sass-lang.com/.

Sencha Touch includes an MVC framework that will allow developers familiar with that pattern to develop applications within a familiar workspace. It also contains a custom URL routing mechanism and history state support.

The widgets that users will interact with in your application comprise the UI portion of Sencha Touch. When thinking about designing and constructing your applications, there is a lot to choose from, which is why I think it's a good idea to look at each of them.

1.3 The Sencha Touch UI

The Sencha Touch UI is a rich mixture of widgets that can be displayed on screen for you and your users to interact with. Given the size of the UI palette, I've organized the following table to help us grasp the groups of UI components.

After reviewing the groups, we'll dive deeper into each group and discuss the UI components in greater detail.

Table 1.2 Looking at the various groups of UI widgets available in the Sencha Touch framework.

Group	Purpose
Containers	Widgets that are designed for nothing more than managing other child items. An example of these types of widgets is the TabPanel. Containers typically implement layouts.
Sheets	Sheets are generally any popup or side-anchored container and appear in a modal fashion, requiring users to interact with the sheet before moving forward. An example of a sheet is the Date Picker widget.
Views	Views are widgets that implement data Stores to display data. The List and Nested List are both views that implement Stores.
Misc	This collection of widgets range from Buttons to Maps to Media.

Now that we have a good overview of the widget groups, we can begin our visual exploration.

1.3.1 Panels

Much like Ext JS, Containers in Sencha Touch do the heavy lifting when it comes to managing widgets inside of widgets. The most commonly used Containers are `Panel`, `TabPanel`, `FormPanel` and `Toolbar`.

Of these, the Panel is the workhorse of Sencha Touch applications as it offers the most when it comes to configurability and flexibility. `Panels` allow other widgets to be placed in either side of the `Panel`, in an area known as a dock.

To see what I mean, take a look at the figure below.

Figure 1.3 A demonstration of some of the Panel content areas.

In Figure 1.3 we see a `Panel` with three docked items. We have a `Toolbar` docked at the top, `List` docked on the left and another toolbar docked at the bottom. Notice how the top-docked `Toolbar` simply contains a title, while the bottom toolbar contains buttons. This shows some of the power and flexibility of the `Toolbars`.

If you have a need to display screens controlled by a toolbar, the `TabPanel` is what you'll need to get the job done.

1.3.2 Controlling your UI with TabPanel

`TabPanel` is a container well and automatically sets a top-docked or bottom-docked toolbar for you with automatically generated buttons for every child item. Tapping any of the buttons allows you to "flip" through items known as "cards". For example, see figure 1.4 below.

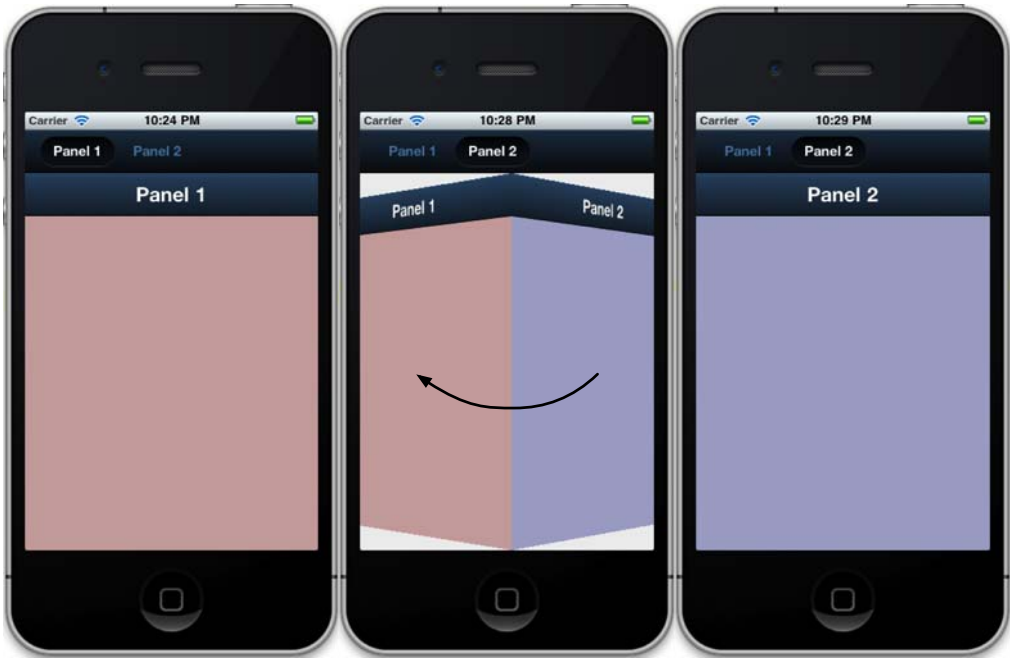

Figure 1.4 The TabPanel allows you to configure UIs that can be changed by a tap of a button and includes optional transition animations (from left to right).

In the figure above, I've configured a `TabPanel` that implements a "cube" transition with two child panels. By selecting Panel 2 in the `Toolbar`, Sencha Touch automatically applies the CSS3 transition properties to both child panel elements, allowing for a smooth transition from one `Panel` to another.

The `TabPanel` does an excellent job of managing the display of items in your screen, but sooner or later, you'll need the ability to accept data input from your users. Here, the FormPanel is what you'll use.

1.3.3 Accepting input with FormPanels

The `FormPanel` is a container that is typically used to display any of the input fields that Sencha Touch provides and is automatically scrollable. Your fields can be grouped via the `FieldSet` widget.

Below is an example of a `FormPanel` requiring user input.

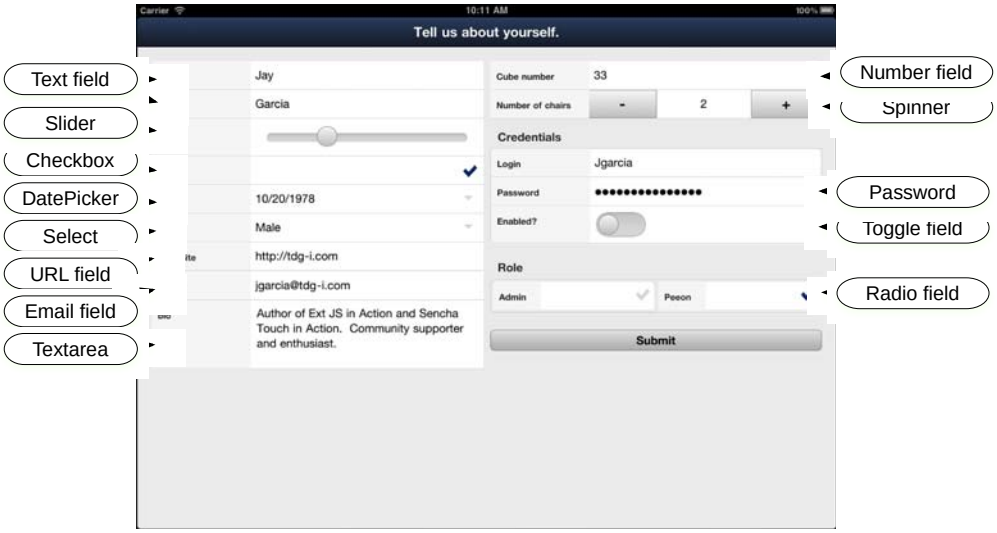

Figure 1.5 FormPanels are used to display input fields and contain necessary controls to manage the submission of data to your server.

In Figure 1.5 (above), we have most of the input fields that Sencha Touch offers, with the exception of the `Hidden` field. With Sencha Touch, the `Text`, `Checkbox`, `URL`, `Email`, `Textarea`, `Number`, `Password` and `Radio` fields all implement native HTML5 input elements, with the addition of styling. Each of these, minus the `Radio` and `Checkbox` fields, will force the native slide in keyboard to appear when focused, allowing users to enter data into the fields.

The Checkbox and Radio fields work similarly to their native-web counterparts, except they are stylized via Sencha Touch's own check icon to mimic native application behavior. In this example, the Role checkboxes, grouped in a `Fieldset`, are `Radio` fields, allowing only one selection in the set.

Next, the `Spinner` field is a custom styled input field, allowing users to enter numeric values, much like the `Number` field, with the addition of easy-to-use decrement and increment buttons on each side of the field.

The `Slider` field implements native Sencha Touch `Draggable` and `Droppable` classes, allowing users to input a numeric value, via a swipe and tap gestures. The `Toggle` field extends `Slider`, allowing users to toggle a field from two values via swipe and tap gestures, much like an on/off toggle switch that you see in various physical devices.

Lastly, the `DatePicker` field and `Select` fields give your users the ability to choose data from a set. The `DatePicker` field implements what's known as a Sheet, which is an overlay panel that slides in from the bottom, allowing the user to select values via vertical swipe or "flick" gestures.

Below is an example of a `DatePicker` displayed in an iPhone.

Figure 1.7 The DatePicker slider in action.

No matter what the device or its orientation, the `DatePicker` field will always display a sheet forcing selection though this modal overlay. At this point, the `DatePicker` widget should seem familiar to you, as it mimics the native iOS Date Picker input widget.

The `Select` field, however, will display different input widgets, based on the device. Below is an example of our implementation of the Select widget in a phone and tablet devices.

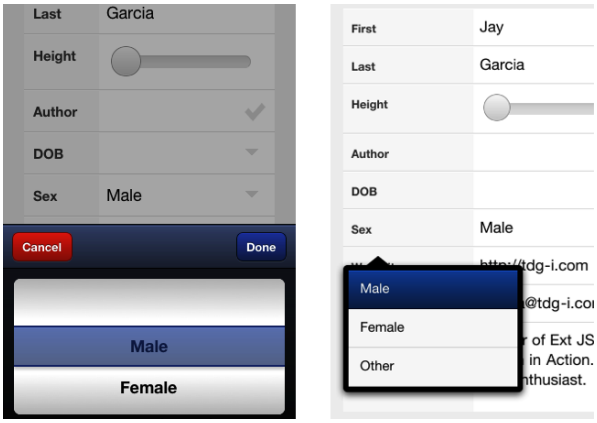

Figure 1.8 The Select field will display a different input widget based on the type of device that is running your

UI.

In the left portion of the illustration above, the `Select` widget is displaying a `Picker` sheet because Sencha Touch detected that it's running inside of a phone versus a tablet (right). The difference has to do w/ the fact that iOS tablets natively display the dialogue-type of controls for selection.

As we've just seen, Sencha Touch offers quite a lot of wrapped native HTML5 input fields, as well as a few custom widgets. Since we've been talking about the DatePicker and Select field implementing Sheets, we should take a look at the various Pickers and Sheets that Sencha Touch offers, outside of the `FormPanel`.

1.3.4 Sheets and Pickers

We've already seen the `DatePicker` and Picker classes implemented via their associated form input widgets. Sencha Touch provides you with a widget called Sheet, which is a floating modal panel that animates into view, grabbing the user's attention and focus.

Below is an example of a `Sheet` in action.

Figure 1.9 A generic Sheet in action displayed in portrait mode.

In the above illustration, we are displaying a Sheet with top and bottom-docked Toolbars, managing a scrollable `List` view. We can configure such a UI because `Sheet` is actually a subclass of `Panel` and brings forth all of the UI goodness that it provides.

What's neat about the `Sheet` widget is that it's orientation aware. This means that flipping the device while the application code above is executing immediately causes the sheet to render in landscape mode, as illustrated below.

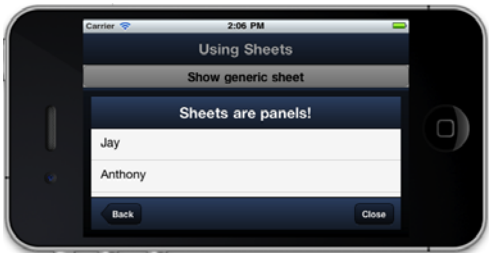

Figure 1.10 A generic Sheet displayed in the landscape mode.

The story about Sheet does not end here. It has three subclasses, which are `ActionSheet`, `MessageBox` and Picker. Now, we've already seen `Picker` and its subclass, `DatePicker`, but we have not seen `ActionSheet` and `MessageBox`.

I know that I'm throwing a lot of new names at you, so to help with any confusion, I've included a simple inheritance model diagram.

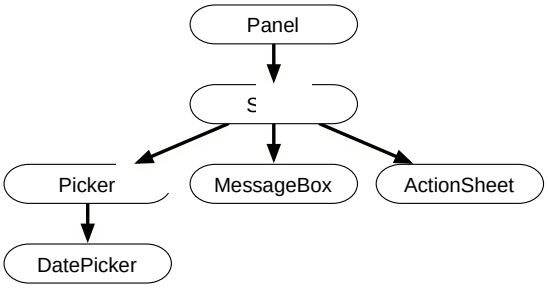

Figure 1.11 The Sheet inheritance model.

Out of the box, the `ActionSheet` widget allows you to easily render buttons in a sheet, rendered in a vertical stack. Because `ActionSheet` is a `Sheet`, which extends `Panel`, we can add pretty much anything we want to the stack of widgets.

Here is an example of an `ActionSheet` rendered with a custom HTML title.

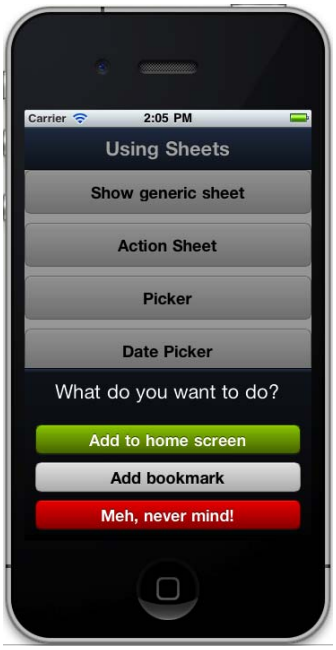

Figure 1.12 An ActionSheet displayed in an application.

Such an `ActionSheet` could be used to request action from the user, requiring that they choose an action via one of the buttons. In this case, we're asking the user to choose one of three options, and with the custom title, are providing a hint along with the actionable button set.

The `MessageBox` widget is a subclass of Sheet that provides Sencha Touch styled alert-like functionality to our applications. Here is an example of the three most common `MessageBox` display patterns.

Figure 1.13 The common MessageBox implementations alert (top left), confirm (bottom left) and prompt (right).

The illustration above displays the three most common uses of MessageBox, including alert (top left), confirm (bottom left) and prompt (top right). Each of these dialogues appear with smooth CSS3 transitions and mimic their native counterparts.

The key difference between the three is apparent. The alert `MessageBox` is designed to alert the user of some condition and only displays one button. The confirm dialog actually allows the user to make a decision by tapping on a button, allowing for a branch of logic to execute. Lastly, the prompt `MessageBox` asks the user for direct input.

We've seen all that Sheet and its subclasses have to offer. Next, we'll look at the various data-bound views that Sencha Touch offers.

1.3.5 Data-bound views

If you're an Ext JS developer, you might be surprised to learn that Sencha Touch only provides three data-bound views and that this list excludes a `GridPanel`. What we have at our disposal is a `DataView`, `List` and `NestedList`. We'll begin with the most basic, and work our way to the most complex, starting with the `DataView`.

The `DataView` is a widget that binds to a data Store to render data on screen. It gives you 100% control on how you will render your data. Below is an example of a simple `DataView` displaying a set of names, beginning with the last name.

Figure 1.14 A stylized DataView with an ”itemtap” event handler, displaying an action sheet based on selection.

Above, we have a stylized `DataView` rendering data from a Store, which contains a list of names. Now, I just rendered names to keep it simple, but `DataViews` can be used to render pretty much anything imaginable, and allow for user interaction.

With the `DataView`, we’re completely responsible for a lot of work, including defining the `XTemplate` that will be used to stamp out the HTML fragments, and styling how the items are rendered on screen. If you’re like me and want the look and feel of a native list, the `List` class is available at your disposal.

Here’s a List widget, rendering the exact same data we have above.

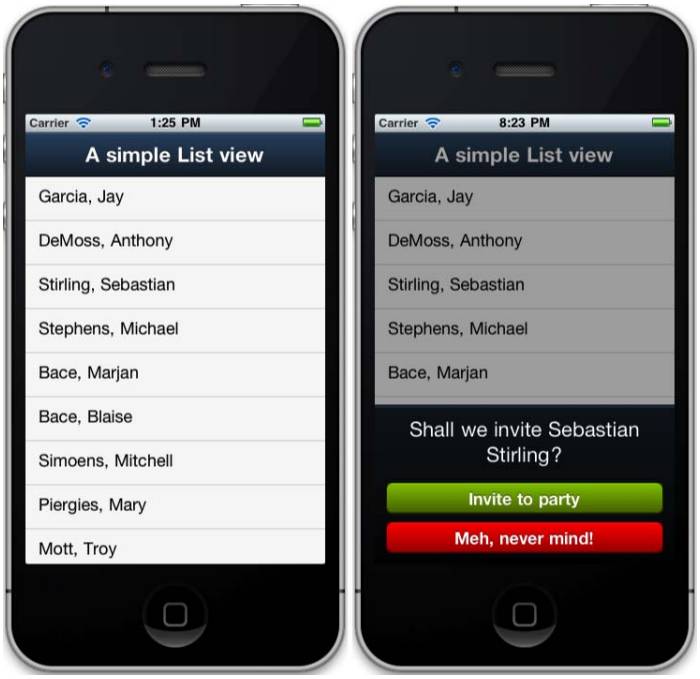

Figure 1.15 A simple ListView with an action sheet.

Illustrated above is a `List` view widget rendering the same data that we did in the `DataView` example we looked at previously. The difference between the two is pretty obvious. What is not so obvious is that the level of effort to create this list view is orders of magnitude less than creating the previously viewed `DataView`. You'll see what I mean by this in Chapter 6, when we tackle List views head on.

There are three more key features that the List view provides in addition to the native application look and feel, and those are illustrated below for further discussion.

Figure 1.16 A grouped list view, sporting an Index bar.

With a few minor tweaks, we are able to transform a simple list view into a grouped `List` view. The grouped List in the illustration above has what is known as a grouping bar, which is a separator between items in the list. The Sencha Team has been able to get this list to work nearly identically to native grouped lists, and includes optional disclosure icons, as well as an index bar, for fast searching with a single finger swipe.

The `DataView` and `List` widgets are designed to display data in a linear set. However, there are times where you want to display nested data. For that, you'll need to use the `NestedList` widget.

Here is a NestedList widget in action.

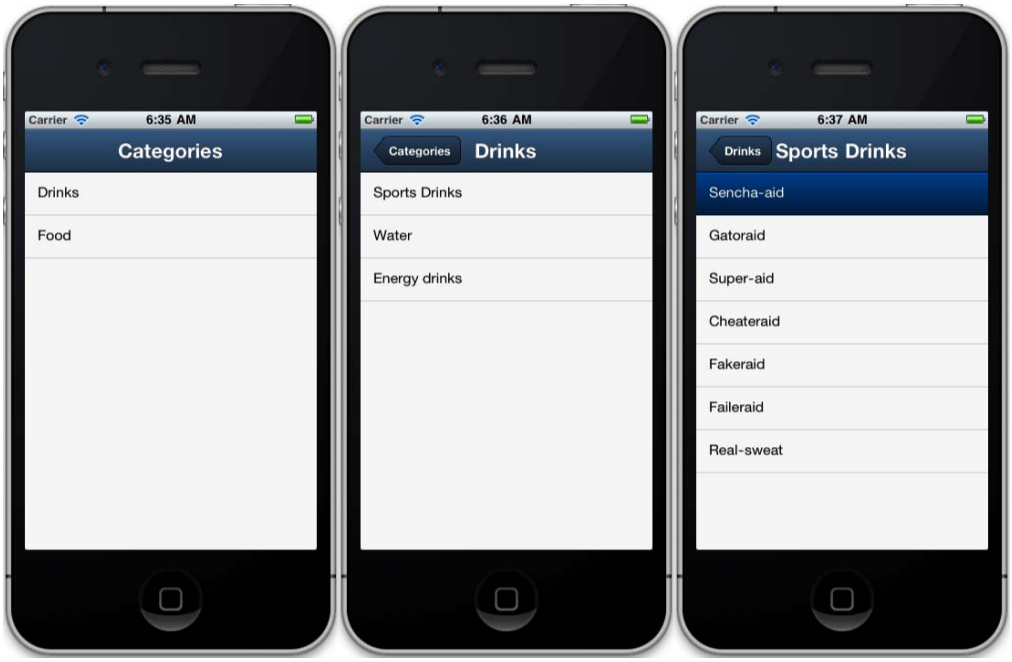

Figure 1.17 A NestedList in action used in a navigational manner.

In the illustration above, I have setup a NestedList for the selection of a food item. There are two main categories, Drinks, and Food. I chose "Drinks" (left), which brought me to three sub-categories. I then chose "Sports Drinks" (center), which led me to the last section of items, which is a list of sports drinks (right). All of this is done w/ the slide animation.

We've just explored the world of the `DataView`, `List` and `NestedList` widgets. Next, we'll explore the world of the Map and Media widgets.

1.3.6 Maps and Media

With the rapid expansion world of mobile applications, having the ability to integrate maps in your applications can be a huge boost in productivity for your users. To meet this growing demand, Sencha Touch integrates Google Maps to supercharge your location-aware applications.

Below is a screenshot of the Sencha Touch `Map` widget in action.

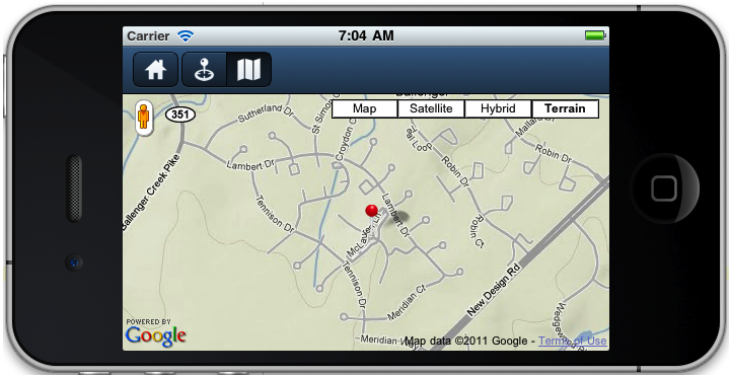

Figure 1.18 The Sencha Touch Map component in landscape mode.

The Sencha Touch `Map` widget literally *wraps* Google Maps, allowing your application code to manage the Google Maps instance as if it was native to Sencha touch. This means that the `Map` widget can take part in layouts, and has normalized events, as well as an interface method to easily update the map's coordinates.

Another growing demand for mobile applications is the ability to play audio and video content. HTML5 natively has video and audio tags that bring this functionality to Mobile Safari, but Sencha Touch makes it easier to use.

Below is an example of the Media widget displaying a video on a tablet.

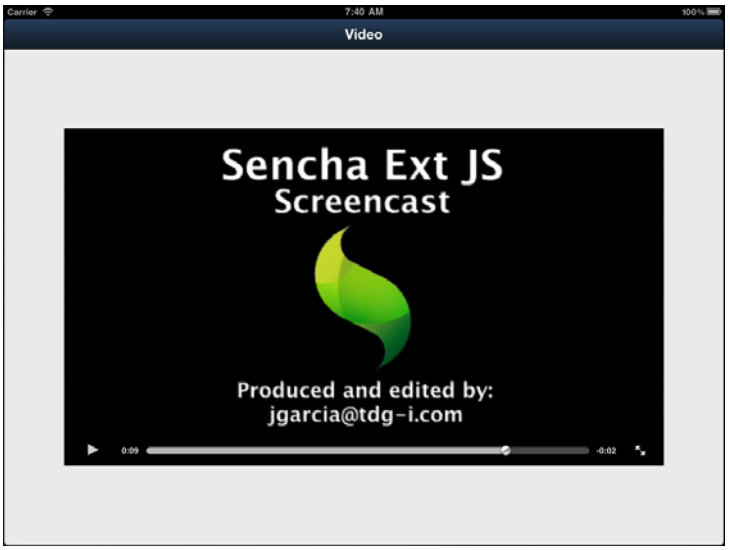

Figure 1.19 A Panel wrapping a Media widget to play video on a tablet.

Just like the Map widget, the Media widget brings Sencha-like interface methods, and easy configurability to play audio and video in your applications.

We've just completed our UI walkthrough. Before we wrap up this chapter, I want to talk to about thinking like a Mobile developer. This conversation will be especially helpful to you if this is your first dive into the world of Mobile application development. I know you're itching to get down to coding, so I won't hold you very long.

1.4 Thinking like a mobile developer

If you're like me, you're making the transition from Ext JS to Sencha Touch. However, making the transition to mobile from desktop application development from a thought-perspective poses challenges that must be overcome if you are to build successful apps.

Even though Sencha Touch is a relatively new framework, there are some things that immediately come to mind when making the transition from desktop to mobile. Here are some points you need to think about before moving forward with your application development.

1.4.1 Think light-weight

When spec'ing out or developing your mobile app, you must think "lightweight" or your app is destined to run into performance issues. If you've made the transformation from a native-desktop application developer to a desktop-web application developer, it is likely that you've hit this issue during that transition, where native desktop applications can handle much more of a burden than desktop-web applications.

Due to the reduced computing power of mobile devices, the mobile browser is limited in many ways, when compared to its desktop counterpart**.** This is why thinking "lightweight" is paramount for a successful application.

My suggestion is to try to reduce the amount of data as well as the complexity of the screen size as possible. Reducing the user interaction models is also a plus, as complex user interaction models for mobile web applications.

1.4.2 Remember - it's a browser!

Many developers are tasked with converting native applications to Sencha Touch-powered web applications. Often times during the conversion process, performance issues are hit and Sencha Touch is blamed.

It is during these times that I tell developers who are caught in this cycle to remember that the application they are developing *is* running inside of a browser, thus has limited power relative to native-compiled mobile applications. Just like native-desktop applications can handle more difficult tasks when compared to desktop-web applications, native-mobile apps have more muscle when compared to mobile-web applications.

I believe that entering the conversion process with this in mind helps you truly set realistic expectations with your customers.

1.4.3 Throw away what you don't need

With the reduced power of mobile devices comes an increase of responsibility to keep things as clean as possible, and reduce DOM clutter/bloat. For desktop-web applications, this is not so critical, but for mobile-web it is extremely critical.

This means that when placing items in the DOM, such as `ActionSheets`, you must take the care to destroy them. DOM bloat is the enemy of performance.

Mobile Safari will crash

> Mobile Safari will crash if your application causes it to run out of memory. This will simply cause the application to disappear from the user, without warning.

Sencha Touch widgets come with a complete three-phase lifecycle, allowing you to easily destroy components, thus removing items from the DOM and freeing up crucial resources.

Along with the destruction of items that are no longer needed is the thought of only instantiating what is truly needed. I often find hugely nested components being instantiated when only a single component is needed for a particular action. To keep things safe, try to think conservatively.

1.4.4 "finger" !== "mouse"

Part of transitioning to mobile development is the understanding the user interaction models and how they relate to browser events. The following table describes some of the most common user gestures, alongside their desktop counterparts (when applicable).

Table 1.2 Comparing touch gesture events with desktop mouse gesture events.

Mobile	Desktop	Description
tapstart	mousedown	The initial point at which a touch is detected in the UI.
tapend	mouseup	Signals the end of a touchstart event.
tap	click	A tapstart and tapend event for a single target.
doubletap	doubleclick	Two successive tapstart and tapends for a single target.
swipe		A tapstart and tapend events with a delta in X (horizontal) or Y (vertical) coordinates.
pinch		A complex multi-finger "pinch" gesture. It is comprised of multiple touchstart and touchend events with deltas in the X and Y coordinate space.

Always try to test all of your complex interaction models with the physical platforms that you are targeting for your applications. It's only then that you can truly see how it will react during events like pinch, swipe and drag.

1.4.5 Reduce the data

When developing your applications, you have to remember to reduce the amount of data you're sending to the browser. If you find yourself pushing megabytes of data to the server for a single Ajax request, you should reconsider your approach.

Along these lines is the reduction in data complexity. Remember that these devices are relatively low powered, and any time spent manipulating complex data could be spent allowing the user to interact with the application. Tasking your mobile application to deal with deeply nested and complicated data structures is highly discouraged.

> **Server side developers will have to work harder**
>
> Lots of times deeply nested data structures are passed to clients because of the amount of work it is for the server side developers to reduce complexity. I'd much rather have our server-side developers work harder than impact the performance of the client, thus shedding negative light on our mobile applications.

Through your application development iterations, I suggest often placing yourself in the shoes of the end-user. Remember, mobile applications should be quick and responsive.

1.5 Summary

We covered quite a bit in this first chapter, beginning with a high-level discussion about what Sencha Touch is and the problems that it aims at solving for the mobile-web application space.

We then took a deep dive into the world of the SenchaTouch UI widget set and got to learn about hat is offered out of the box. Along the way, we identified some of the differences between widgets, such as the DataView and List view.

Lastly, we discussed some of the ways you should think about mobile applications and some of the limitations that mobile devices pose.

In the next chapter, we're going to begin our deep dive into Sencha Touch, beginning with where to get the framework, and inspect its contents. After we've become familiar with setting up a basic Sencha Touch app page, we'll develop a quick application with the framework.