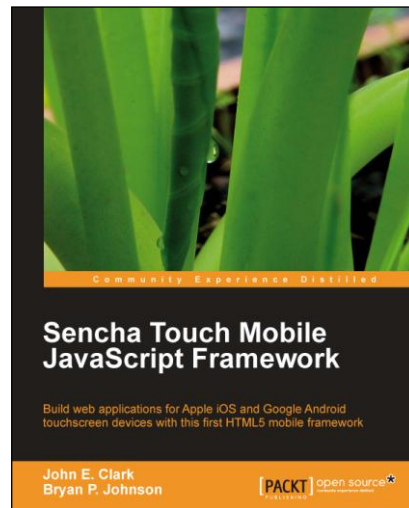


# Sencha Touch Mobile JavaScript Framework

**John E. Clark**  
**Bryan P. Johnson**



## Chapter No. 2 "Creating a Simple Application"

## In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.2 "Creating a Simple Application"

A synopsis of the book's content

Information on where to buy this book

## About the Authors

**John E. Clark** holds a Master's Degree in Human-Computer Interaction from Georgia Tech and an undergraduate degree in Music Engineering from Georgia State University. John and his co-author, Bryan Johnson, worked together at MindSpring, and later EarthLink, starting out in technical support and documentation, before moving into application development, and finally, management of a small development team. After leaving Earthlink in 2002, John began working independently as a consultant and programmer, before starting Twelve Foot Guru, LLC, with Bryan, in 2005.

John has been working with Sencha Touch since the first early beta releases. He has also worked with Sencha's ExtJS since the early days, when it was still known as YUI-Ext.

When he is not buried in code, John spends his time woodworking, playing guitar, and brewing his own beer.

---

I would like to thank my family for all of their love and support. I would also like to thank Bryan for his help, his patience, and his continued faith in our efforts.

---

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)

**Bryan P. Johnson** is a graduate of the University of Georgia. Bryan went to work for MindSpring Enterprises in late 1995, where he met his co-author John Clark. At MindSpring, and later Earthlink, for over seven years, Bryan served in multiple positions, including Director of System Administration and Director of Internal Application Development. After leaving Earthlink, Bryan took some time off to travel before joining John in starting Twelve Foot Guru.

Bryan has worked with Sencha's products since the early days of YUI-Ext and has used Sencha Touch since its first betas.

---

I would like to thank my family for their support and my co-author John for his patience during the creation of this book.

---

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)

# Sencha Touch Mobile

## JavaScript Framework

Since its initial launch, Sencha Touch has quickly become the gold standard for developing rich mobile web applications with HTML5. Sencha Touch is the first HTML5 mobile JavaScript framework that allows you to develop mobile web applications that look and feel like native applications on both iPhone and Android touchscreen devices. Sencha Touch is the world's first application framework built specifically to leverage HTML5, CSS3, and JavaScript for the highest level of power, flexibility, and optimization. It makes specific use of HTML5 to deliver components such as audio and video, as well as a localStorage proxy for saving data offline. Sencha Touch also makes extensive use of CSS3 in its components and themes, to provide an incredibly robust styling layer, giving you total control over the look of your application.

Sencha Touch enables you to design both Apple iOS and Google Android applications without the need for learning multiple arcane programming languages. Instead, you can leverage your existing knowledge of HTML and CSS to quickly create rich web applications for mobile devices in JavaScript. This book will show you how you can use Sencha Touch to efficiently produce attractive, exciting, and easy-to-use web applications that keep your visitors coming back for more.

Sencha Touch Mobile JavaScript Framework teaches you all you need to get started with Sencha Touch and build awesome mobile web applications. Beginning with an overview of Sencha Touch, this book will guide you through creating a complete simple application, followed by styling the user interface and the list of Sencha Touch components, which are explained through comprehensive examples. Next, you will learn about the essential touch and component events, which will help you create rich dynamic animations. The book follows this up with information about core data packages and how to deal with data, and wraps it up with building another simple but powerful Sencha Touch application.

In short, this book has the step-by-step approach and extensive content to turn a beginner to Sencha Touch into an ace, quickly and easily.

Exploit Sencha Touch, a cross-platform library aimed at next generation, touch-enabled devices.

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)

## What This Book Covers

*Chapter 1, Let's Begin with Sencha Touch!:* This chapter provides an overview of Sencha Touch and a walkthrough of the basics of setting up the library for development. We will also talk about programming frameworks and how they can help you develop touch-friendly applications quickly and easily.

*Chapter 2, Creating a Simple Application:* This chapter starts out by creating a simple application and using it to discover the basic elements of Sencha Touch. We will also explore some of the more common components, such as lists and panels, and we will show you how to find common errors and fix them when they occur.

*Chapter 3, Styling the User Interface:* Once we have our simple application, we will explore ways to change the look and feel of individual components, using CSS styles. Then, we will dive into using Sencha Touch themes to control the look of your entire application, using SASS and Compass.

*Chapter 4, Components and Configurations:* Here, we will examine the individual components for Sencha Touch in greater detail. We will also cover the use of layouts in each component, and how they are used to arrange the different pieces of your application.

*Chapter 5, Events:* Following our look at the individual components, we will take a look at the Sencha Touch events system, which allows these components to respond to the user's touch and communicate with each other. We will cover the use of listeners and handlers, and explore ways to monitor and observe events, so that we can see what each part of our application is doing.

*Chapter 6, Getting Data In:* Data is a critical part of any application. Here, we will look at the different methods for getting data into our application, using forms to gather information from the user, and ways to verify and store the data. We will also talk about the different data formats that are used by Sencha Touch and how we can manipulate that data using Sencha Touch's models and stores.

*Chapter 7, Getting Data Out:* Once we have data in our application, we need to be able to get it back out for display to the user. Here, we will discuss the use of panels and xTemplates to display the data. We will also take a look at using our data to create colorful charts and graphs, using Sencha Touch Charts.

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)

*Chapter 8, The Flickr Finder Application:* Using the information we have learned about Sencha Touch, we will create a more complex application that grabs photos from Flickr, based on our current location. We will also use this as an opportunity to talk about best practices for structuring your application and its files.

*Chapter 9, Advanced Topics:* For our final chapter, we will explore ways to synchronize your data with a database server by creating your own API. Additionally, we will look at ways to synchronize data between the mobile device and a database server, compiling your application with PhoneGap and NimbleKit, as well as ways to get started as an Apple iOS or Google Android developer.

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)

# 2

## Creating a Simple Application

This chapter will walk you through creating a simple application in Sencha Touch. We will cover the basic elements that are included in any Sencha Touch application, and we will take a look at the more common components you might use in your own applications: containers, panels, lists, toolbars, and buttons.

In this chapter, we will cover:

- Setting up your folder structure
- Starting from scratch with TouchStart.js
- Controlling the container using layouts
- Testing and debugging the application
- Updating the application for production
- Putting the application into production

Next, we will cover how to use the various containers to display text and other items. We will then add additional components to create our first simple application. Finally, we will take a look at debugging your application and give you some pointers on what to do when things go boom.

### Setting up your folder structure

Before we get started, you need to be sure that you've set up your development environment properly, as outlined in the previous chapter.

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)



### Root folder

As noted in the previous chapter, you will need to have the folders and files for your application located in the correct web server folder, on your local machine.

On the Mac, this will be the `Sites` folder in your Home folder.

On Windows, this will be `C:\xampp\htdocs` (assuming you installed **xampp**, as described in the previous chapter).

Through the rest of the book, we will refer to this folder as the **root folder** of your local web server.

## Setting up your application folder

Before we can start writing code, we have to perform some initial set up, copying in a few necessary resources and creating the basic structure of our application folder. This section will walk you through the basic setup for the Sencha Touch files, creating your style sheets folder, and creating the `index.html` file.

1. Locate the Sencha Touch folder you downloaded in the previous chapter.



The code in this chapter was written using Sencha Touch 1.1.0.

2. Create a folder in the root folder of your local web server. You may name it whatever you like. I have used the folder name `TouchStart` in this chapter.
3. Create three empty sub folders called `lib`, `app`, and `css` in your `TouchStart` folder.
4. Now, copy the `resources` and `src` folders, from the Sencha Touch folder you downloaded earlier, into the `TouchStart/lib` folder.
5. Copy the following files from your Sencha Touch folder to your `TouchStart/lib` folder:
  - `sencha-touch.js`
  - `sencha-touch-debug.js`
  - `sencha-touch-debug-w-comments.js`



6. Create an empty file in the `TouchStart/css` folder called `TouchStart.css`. This is where we will put custom styles for our application.
7. Create an empty `index.html` file in the main `TouchStart` folder. We will flesh this out in the next section.

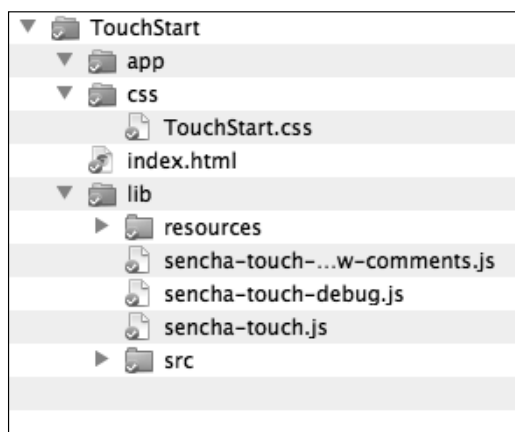
### Icon files

Both iOS and Android applications use image icon files for display. This creates pretty rounded launch buttons, found on most touch-style applications.



If you are planning on sharing your application, you should also create PNG image files for the launch image and application icon. Generally, there are two launch images, one with a resolution of 320 x 460 px, for iPhones, and one at 768 x 1004 px, for iPads. The application icon should be 72 x 72 px. See Apple's *iOS Human Interface Guidelines* for specifics, at <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/IconsImages/IconsImages.html>.

When you're done, your folder structure should look as follows:



## Creating the HTML application file

Using your favorite HTML editor, open the `index.html` file we created when we were setting up our application folder. This HTML file is where you specify links to the other files we will need in order to run our application.

The following code sample shows how the HTML should look:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
    <title>TouchStart Application - My Sample App</title>

    <!-- Sencha Touch CSS -->
    <link rel="stylesheet" href="lib/resources/css/sencha-touch.css"
type="text/css">

    <!-- Sencha Touch JS -->
    <script type="text/javascript" src="lib/sencha-touch-debug.js"></
script>

    <!-- Application JS -->
    <script type="text/javascript" src="app/TouchStart.js"></script>

    <!-- Custom CSS -->
    <link rel="stylesheet" href="css/TouchStart.css" type="text/css">

  </head>
  <body></body>
</html>
```



### Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.



### Comments

In HTML, anything between `<!--` and `-->` is a comment, and it will not be displayed in the browser. These comments are to tell you what is going on in the file. It's a very good idea to add comments into your own files, in case you need to come back later and make changes.

Let's take a look at this HTML code piece-by-piece, to see what is going on in this file.

The first five lines are just the basic set-up lines for a typical web page:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
    <title>TouchStart Application - Hello World</title>
```


With the exception of the last line containing the title, you should not need to change this code for any of your applications. The title line should contain the title of your application. In this case, TouchStart Application - Hello World is our title.

The next few lines are where we begin loading the files to create our application, starting with the Sencha Touch files.

The first file is the default CSS file for the Sencha Touch library — `sencha-touch.css`.

```
<link rel="stylesheet" href="lib/resources/css/ext-touch.css"
type="text/css">
```

[



**CSS files**

**CSS or Cascading Style Sheet** files contain style information for the page, such as which items are bold or italic, which font sizes to use, and where items are positioned in the display.

]

The Sencha Touch style library is very large and complex. It controls the default display of every single component in Sencha Touch. It should not be edited directly.

The next file is the actual Sencha Touch JavaScript library. During development and testing, we use the debug version of the Sencha Touch library, `sencha-touch-debug.js`:

```
<script type="text/javascript" src="lib/sencha-touch-debug.js"></script>
```

The debug version of the library is not compressed and contains comments and documentation. This can be helpful if an error occurs, as it allows you to see exactly where in the library the error occurred.

When you have completed your development and testing, you should edit this line to use `sencha-touch.js` instead. This alternate file is the version of the library that is optimized for production environments and takes less bandwidth and memory to use; but, it has no comments and is very hard to read.

Neither the `sencha-touch-debug.js` nor the `sencha-touch.js` files should ever be edited directly.

The next two lines are where we begin to include our own application files. The names of these files are totally arbitrary, as long as they match the name of the files you create later, in the next section of this chapter. It's usually a good idea to name the file the same as your application name, but that is entirely up to you. In this case, our files are named `TouchStart.js` and `TouchStart.css`.

```
<script type="text/javascript" src="app/TouchStart.js"></script>
```

This first file, `TouchStart.js`, is the file that will contain our JavaScript application code.

The last file we need to include is our own custom CSS file, called `TouchStart.css`. This file will contain any style information we need for our application. It can also be used to override some of the existing Sencha Touch CSS styles.

```
<link rel="stylesheet" href="resources/css/TouchStart.css"
type="text/css">
```

This closes out the `</head>` area of the `index.html` file. The rest of the `index.html` file contains the `<body></body>` tags and the closing `</html>` tag.

If you have any experience with traditional web pages, it may seem a bit odd to have empty `<body></body>` tags, in this fashion. In a traditional web page, this is where all the information for display would normally go.

For our Sencha Touch application, the JavaScript we create will populate this area automatically. No further content is needed in the `index.html` file, and all of our code will live in our `TouchStart.js` file.

So, without further delay, let's write some code!

## Starting from scratch with TouchStart.js

Let's start by opening the `TouchStart.js` file and adding the following:

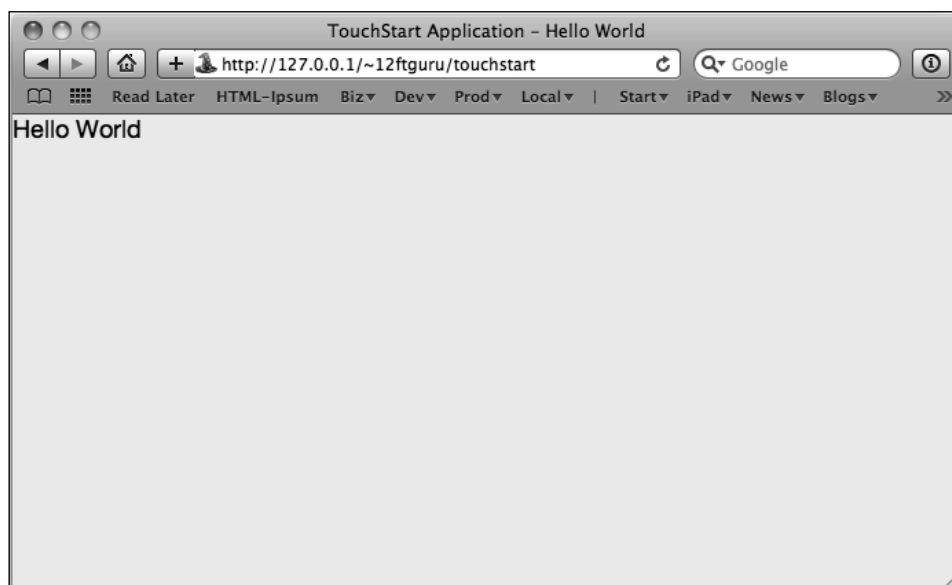
```
new Ext.Application({
  name: 'TouchStart',
  launch: function() {
```

```
var hello = new Ext.Container({
  fullscreen: true,
  html: '<div id="hello">Hello World</div>'
});

this.viewport = hello;
}
```

This is probably the most basic application you can possibly create: the ubiquitous "Hello World" application. Once you have saved the code, use the Safari web browser to navigate to the TouchStart folder in the root folder of your local web server. The address should look like the following:

- `http://localhost/TouchStart/`, on the PC
- `http://127.0.0.1/~username/TouchStart`, on the Mac (username should be replaced with the username for your Mac)



As you can see, all that this bit of code does is create a single window with the words **Hello World**. However, there are a few important elements to note in this example.

The first line, `NewExt.Application({`, creates a new application for Sencha Touch. Everything listed between the curly braces is a configuration option of this new application. While there are a number of configuration options for an application, most consist of at least the application's name and a `launch` function.

### Namespace

One of the biggest problems with using someone else's code is the issue of naming. For example, if the framework you are using has an object called "Application", and you create your own object called "Application", the two functions will conflict. JavaScript uses the concept of namespaces to keep these conflicts from happening.



In this case, Sencha Touch uses the namespace `Ext`. You will see this namespace used throughout the code in this book. It is simply a way to eliminate potential conflicts between the frameworks' objects and code, and your own objects and code.

Sencha will automatically set up a namespace for your own code as part of the new `Ext.Application` object.

`Ext` is also part of the name of Sencha's web application framework called `ExtJS`. Sencha Touch uses the same namespace convention to allow developers familiar with one library to easily understand the other.

When we create a new application, we need to pass it some configuration options. This will tell the application how to look and what to do. These configuration options are contained within the curly braces (`{ }`) and separated by commas. The first option is as follows:

```
name: 'TouchStart'
```

This sets the name of our application to whatever is between the quotes. This name value should not contain spaces, as Sencha also uses this value to create a namespace for your own code objects. In this case, we have called the application `TouchStart`. The next option is where things start to get interesting:

```
launch: function() {  
  var hello = new Ext.Container({  
    fullscreen: true,  
    html: '<div id="hello">Hello World</div>'  
  });  
  
  this.viewport = hello;  
}
```

The `launch` configuration option is actually a function that will tell the application what to do once it starts up. Let's start backwards on this last bit of code for the `launch` configuration and explain `this.viewport`.

By default, a new application has a viewport. The viewport is a pseudo-container for your application. It's where you will add everything else for your application. Typically, this viewport will be set to a particular kind of container object.

At the beginning of the `launch` function, we start out by creating a basic container, which we call `hello`:

```
var hello = new Ext.Container({
  fullscreen: true,
  html: '<div id="hello">Hello World</div>'
});
```

Like the `Application` class, a new `Ext.Container` class is passed a configuration object consisting of a set of configuration options, contained within the curly braces (`{}`) and separated by commas. The `Container` object has over 40 different configuration options, but for this simple example, we only use two:

- `fullscreen` sets the size of the container to fill the entire screen (no matter which device is being used).
- `html` sets the content of the container itself. As the name implies, this can be a string containing either HTML or plain text.

Admittedly, this is a very basic application, without much in the way of style. Let's add something extra using the container's layout configuration option.

#### **My application didn't work!**



When you are writing code, it is an absolute certainty that you will, at some point, encounter errors. Even a simple error can cause your application to behave in a number of interesting and aggravating ways. When this happens, it is important to keep in mind the following:

- Don't Panic.
- Retrace your steps and use the tools mentioned in the previous chapter to track down the error and fix it. If anything from this chapter does not work for you, jump to the *Testing and debugging* section of this chapter for some pointers on where to start looking.

## Controlling the container with layout

Layouts give you a number of options for arranging content inside containers. Sencha Touch offers four basic layouts for containers:

- `fit`: A single item layout that automatically expands to take up the whole container
- `hbox`: Arranges items horizontally in the container
- `vbox`: Arranges items vertically in the container
- `card`: Arranges items like a stack of cards where only the active card is initially visible

In our previous example, we did not declare a layout. In general, you will always want to declare a layout for any container. If you don't, the components inside the container may not size themselves appropriately when they appear. This is not as critical when the container only contains HTML.

Let's take our previous example and modify it a bit:

```
new Ext.Application({
  name: 'TouchStart',
  launch: function() {
    var hello = new Ext.Container({
      fullscreen: true,
      layout: {
        type: 'vbox',
        align: 'stretch'
      },
      items: [
        {
          xtype: 'container',
          flex: 2,
          html: '<div id="hello">Hello World Top</div>',
          cls: 'blueBox',
          border: 1
        }, {
          xtype: 'container',
          flex: 1,
          html: '<div id="hello">Hello World Bottom</div>',
          cls: 'redBox',
          border: 1
        }, {
          xtype: 'container',
          height: 50,
```



```

    html: '<div id="footer">Footer</div>',
    cls: 'greenBox'
  }

  ]
  });

  this.viewport = hello;
}
});

```

For this example, we have removed the line that previously set HTML, '`<div id="hello">Hello World</div>`', and replaced it with our layout configuration:

```

  layout: {
    type: 'vbox',
    align: 'stretch'
  }

```

This configuration sets our main container layout to `vbox` (objects aligned vertically) and stretches the boxes to take up the full horizontal width on the screen.

We have also added `items` to our container, after we set the layout. The items are a collection of Sencha Touch components we want to include inside our container. The `items` list is enclosed in brackets, and the individual components within the `items` list are contained in curly braces.

In this case, we are going to include three additional containers inside of our main container:

```

  items: [
    {
      xtype: 'container',
      flex: 2,
      html: '<div id="hello">Hello World Top</div>',
      cls: 'blueBox',
      border: 1
    }, {
      xtype: 'container',
      flex: 1,
      html: '<div id="hello">Hello World Bottom</div>',
      cls: 'redBox',
      border: 1
    }, {
      xtype: 'container',

```

```
height: 50,  
html: '<div id="footer">Footer</div>',  
cls: 'greenBox'  
}  
  
]
```

One of the first things you will notice is the addition of a configuration called an `xtype`.

In Sencha Touch, `xtype: 'container'` is just another way of saying `new Ext.Container`. This is a much easier way to add items within an existing container and has the added benefit of saving device memory. Almost every component used in Sencha Touch has a unique `xtype`.



#### **xtypes**

When you use `xtype`, the component isn't created until it is actually needed for display to the user. By contrast, any time you use the `new` command, such as `new Ext.Container`, the component is created in memory immediately.

The Sencha Touch API contains a list of the available `xtype` components at <http://dev.sencha.com/deploy/touch/docs/> (search for *Component*).

The next configuration in the list is called `flex`. The `flex` configuration is unique to the `hbox` and `vbox` layouts. It controls how much space the component will take up, proportionally, in the overall layout. You may also have noticed that the last container does not have a `flex` configuration. Instead, it has `height: 80`. The `vbox` layout will interpret these values to lay out the container as follows:

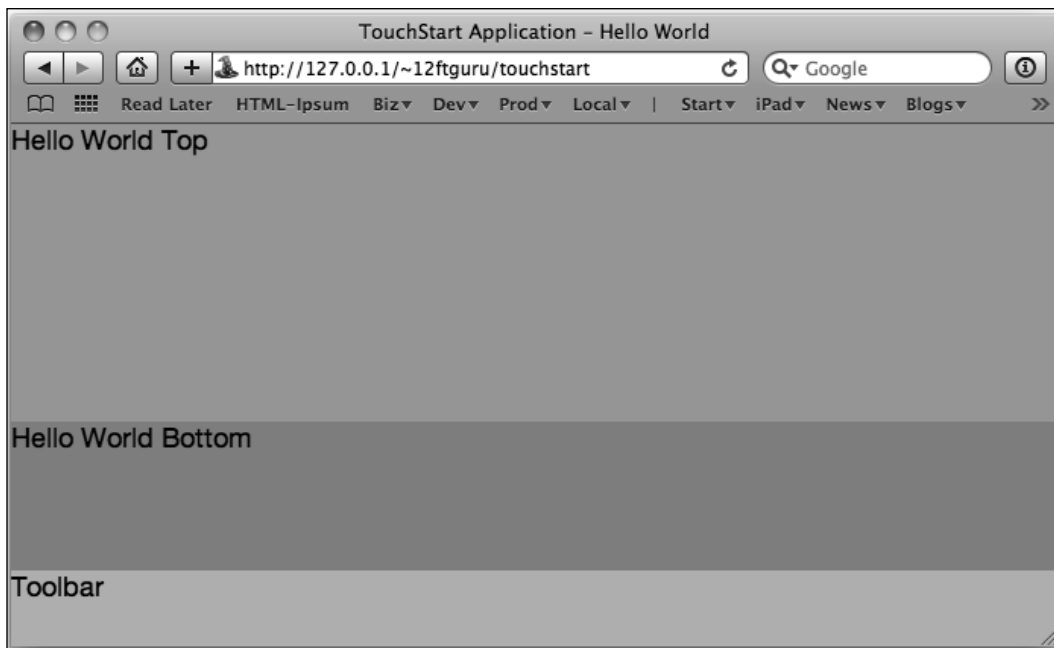
1. Since we have one component with a height of 50, the `vbox` layout will leave that component as 50 pixels tall.
2. The `vbox` layout will then use the `flex` values of the other two components as a ratio. In this case, 2:1.
3. The end result is a container, 80 pixels high, on the bottom of the screen. The other two containers will take up the rest of the available space. The top container will also be twice as tall as the middle container.

In order to make these sizes clearer, we have also added a `cls` configuration to each of the inner containers. The `cls` configuration sets a CSS class on the container, allowing us to use our `TouchStart.css` file to add style changes for each of the containers.

Locate your `TouchStart.css` file, open it in your code editor, and add the following:

```
.blueBox {  
  background-color: #7FADCF;  
}  
  
.redBox {  
  background-color: #CE7E83;  
}  
  
.greenBox {  
  background-color: #7ECEA0;  
}
```

Save your changes and reload the page in Safari:



As you can see from this example, we can easily nest containers to create more complex layouts. We should also take into account the fact that each of these containers can have a different layout and can contain its own items. It would be easy enough to add buttons to our footer container on the bottom and make it into a real working toolbar. However, we really don't need to, because Sencha Touch has already provided us with a simpler way.



An important concept to understand when working with layouts is that the layout configuration does not change where the container itself lives, or how it looks, but only affects the items inside the container. Additionally, all display components in Sencha Touch have a default layout of `fit`. If you don't specify a different layout type, a `fit` layout will be used. We will cover layouts in depth, later in the book.

## The panel

As we noted previously, containers are a very basic object in Sencha Touch, and they can be extended to create more complex objects with new features. The first of these is `Panel`.

Like the container, a panel can have a `layout`, `html`, or `items` components and it can be set to `fullscreen`. In fact, since it inherits from the container, it can do everything a container can, and more.

One of the key advantages of a panel is the ability to have docked items. These docked items can be used to make title bars, toolbars, and navigation bars. A simple example would be the following:

```
new Ext.Application({
    name: 'TouchStart',
    launch: function() {

        this.viewport = new Ext.Panel({
            fullscreen: true,
            bodyPadding: 5,
            dockedItems: [
                {
                    dock: 'top',
                    xtype: 'toolbar',
                    title: 'Touch Start'
                },
                {
                    dock: 'top',
                    xtype: 'toolbar',
                    items: [
                        {
                            text: 'Hello Button'
                        }
                    ]
                }
            ]
        })
    }
})
```

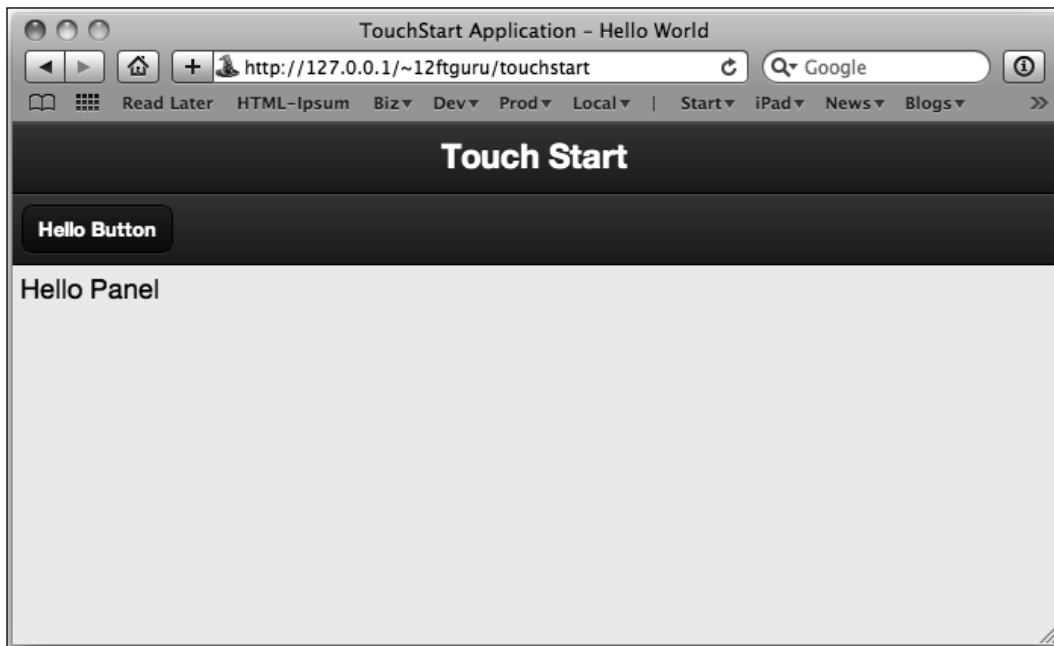
```

    }
  ],

  html: 'Hello Panel'

  });
}
});

```



This is similar to our first example, except we are now using a panel and we have also taken a shortcut by directly setting the `viewport` method:

```
this.viewport = new Ext.Panel({
```

We also added a bit of visual appeal, in the form of padding, to the panel, by setting `bodyPadding` to 5, but the big change is the docked items.

Much like when we added containers into our main container, the docked items are shown as an array of components inside brackets. In this case, there are two items: one for the title bar and a second for the button bar. Both of these items have an `xtype` value `toolbar`.

The `toolbar` also inherits from our old friend, the `container`. This means it can also have `layout`, `html`, and `items` components. As a `dockedItems` component, `toolbar` also understands the concept of where it should be docked. The `dock` configuration can be set to `top`, `right`, `left`, or `bottom`.

Our first toolbar simply sets a `title` configuration value instead of `html` or `items`. However, the second toolbar is a bit different.

One of the interesting things you might have noticed in the second toolbar is that our **Hello Button** doesn't actually have a configuration value for `xtype`. This is because the toolbar assumes that all of its items are buttons, unless you tell it otherwise. In this case, the button only has a `text` property. At this point, the button doesn't do anything, but you can begin to see some of the possibilities for `panel` and its `dockedItems` component.

It should also be noted that `dockedItems` components don't have to be toolbars. They can actually be any kind of component you like. For example, if you want to have a left sidebar, you could add a `dockedItem` component with an `xtype` value of `Panel` and a `dock` setting of `left`, which would give you all the functionality of a regular panel, pinned to the left side of your existing panel.



#### When do I use a panel instead of a container?

Since the panel does so much more than a container, the logical question would be: why use the container at all?

The general rule of thumb is, if you need the extra functionality, use the `panel` component, and if you don't require docked items or a title bar, use the `container` component. Using the `container` saves a bit of memory and makes for cleaner, more understandable code.

While the `panel` is a good starting point for starting an application, Sencha Touch also provides a more complex version of the panel called the `TabPanel`.

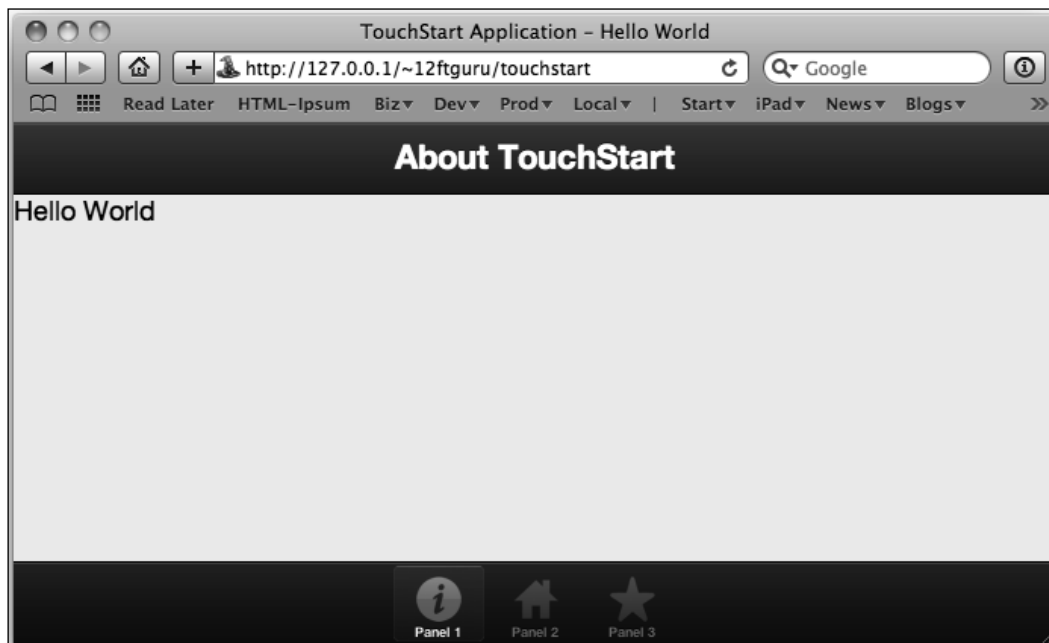
## The TabPanel component

`TabPanel` components contain all of the core functions of the regular panel, but they have a few extra advantages as well.

`TabPanel` is a very specialized panel that uses a card layout to quickly create a switchable set of tabs for each item in the tab panel.

The `card` layout can contain a number of different containers, but it only displays them one at a time. Much like a deck of cards where only the top card is visible, the `card` layout can only have one active item at a time. You can then change the active item and the new card will automatically switch to the front, hiding the previous card.

The `TabPanel` creates a `card` layout for all of its items by default. It also adds a button in the tab bar for each of its items. These buttons automatically switch from one card to the next, without any additional code.



To create this new `TabPanel`, let's modify our previous example. Instead of setting the `viewport` method to a simple panel, we will set it to be a `TabPanel` and put our old panel inside `TabPanel`. We will also add two empty containers, so we can see how the tabs work:

```
new Ext.Application({
  name: 'TouchStart',
  launch: function() {

    this.viewport = new Ext.TabPanel({
      fullscreen: true,
      cardSwitchAnimation: 'slide',
      tabBar: {
```

```
dock: 'bottom',
layout: {
  pack: 'center'
},
items: [{
  xtype: 'panel',
  title: 'Panel 1',
  fullscreen: false,
  html: '<div id="hello">Hello World</div>',
  iconCls: 'info',
  dockedItems: [
    {
      dock: 'top',
      xtype: 'toolbar',
      title: 'About TouchStart'
    }
  ], {
    xtype: 'container',
    html: 'TouchStart container 2',
    iconCls: 'home',
    title: 'Panel 2'
  }, {
    xtype: 'container',
    html: 'TouchStart container 3',
    iconCls: 'favorites',
    title: 'Panel 3'
  }]
});
```

Looking at the code, you can see that `TabPanel` has a couple of new configuration options. The first is the `cardSwitchAnimation` option, which we have set to `slide`. Other options include:


- `fade`
- `flip`
- `cube`
- `pop`
- `wipe`



You can also set this to `false`, which simply swaps the cards without any animation.

`TabPanel` also has a `tabBar` property that functions in much the same way as the `toolbar` component from our previous examples. In this example, we have set `tabBar` to appear at the bottom, and we have set the `tabBar`'s layout to place all of the buttons together (`pack`) in the middle (`center`) of the `tabBar`.

For the `TabPanel` `items` list, we have our original Hello World `panel` component and a pair of simple `container` components. One difference you will see with these items is that we now have configuration options for `title` and `iconCls`. These two options control what appears on the tab for the item. `iconCls` can be set to one of the included icons, or you can customize and include your own icons.



**Icons**

A full list of the available icons can be found in the **Kitchen Sink** application (<http://dev.sencha.com/deploy/touch/examples/kitchensink/>) under **Interface | Icons**. Click on the **Source** button to see how the icons are used.

Now that we have our `TabPanel` component, load the application in Safari and click through the tabs to see how they work. Change the `cardSwitchAnimation` option and see what the other options look like. You can also try changing some of the values for `iconCls`. When you are ready to move on, we will add something a bit more interesting and complex to our `TabPanel`.

## The list component

The `list` component in Sencha Touch allows you to display data in a list layout. This seems pretty straightforward, but the list follows a slightly different pattern than the previous components. Let's make some modifications to our current code and see what some of those differences look like.

Find the set of parentheses that contain the first of our empty containers:

```
{
  xtype: 'container',
  html: 'TouchStart container 2',
  iconCls: 'home',
  title: 'Panel 2'
}
```

Replace that entire `container` component (including the `{}` at either end) with the following:

```
{
  xtype: 'list',
  title: 'List',
  fullscreen: false,
  iconCls: 'bookmarks',
  itemTpl: '{id} - {fullname}',
  store: new Ext.data.Store({
    model: 'ListItem',
    data: [
      {id: 1, fullname: 'Aaron Karp'},
      {id: 2, fullname: 'Baron Chandler'},
      {id: 3, fullname: 'Bryan Johnson'},
      {id: 4, fullname: 'David Evans'},
      {id: 5, fullname: 'John Clark'},
      {id: 6, fullname: 'Norbert Taylor'}
    ]
  })
}
```

The first big difference we can see is that the `list` component does not declare a layout. Instead, it uses an `itemTpl` object to control how the items within the list are arranged. Notice that the elements in curly braces, `'{id} - {fullname}'`, also appear in our `data` component, at the bottom of the `list` component. This means that each row of the list will appear with the ID, a dash, and the value.

These `itemTpl` layout values are called **XTemplates** in Sencha Touch. The XTemplates consists of a string with items in curly braces. When the list appears, it will print out the XTemplates and substitute the items inside the `{}` with the corresponding value listed in the `data`.

`store` is used to control the data for the list. It can keep the data locally, as in our example, or it can retrieve the data from a server. The data within the store has to conform to a **model**. The model describes what values are available to the store and any special attributes they may have.

We will cover XTemplates, stores, and models in greater detail, later on in this book. For now, we still need to create the actual model this store example needs. Right now, the `model` configuration option for the `store` component is set to `ListItem`.

At the top of the application, create a new line right after the `launch` function:

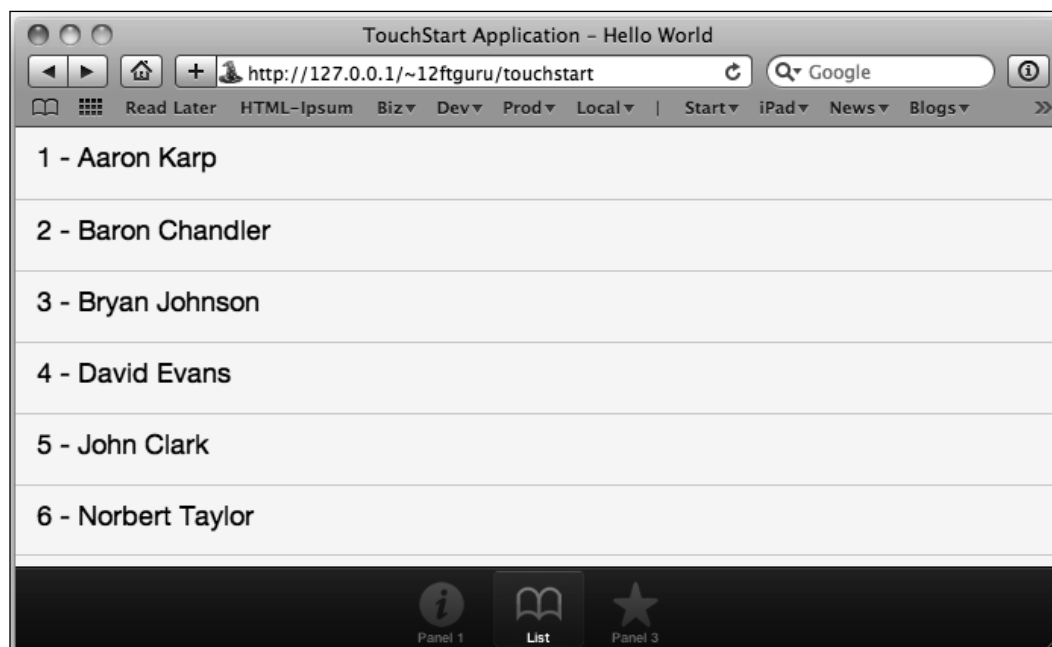
```
launch: function() {
```

Add the following code:

```
Ext.regModel('ListItem', {  
  fields: [  
    {name: 'id', type: 'int'},  
    {name: 'fullname', type: 'string'}  
  ]  
});
```

This will create the correct model for our store. The model creates an array of fields, each of which has a name and a type. The name should match the one you use in the `itemTpl` object and the `data` object for the store. The `type` configuration option lets the store understand how to deal with the data when it is sorted or stored.

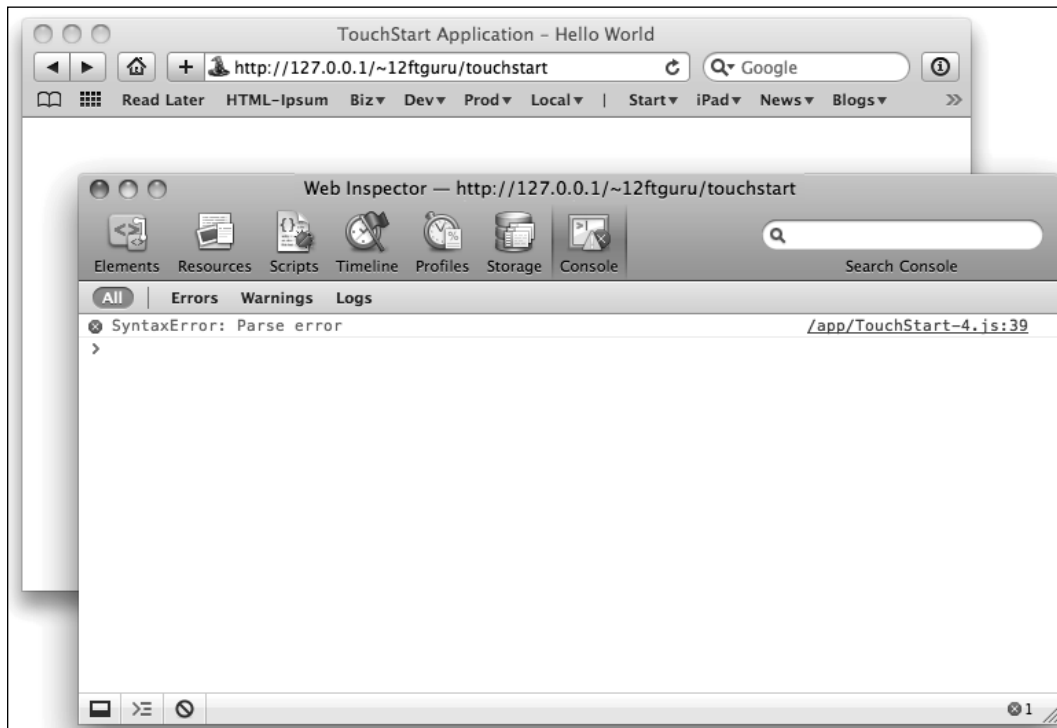
If everything has gone as planned, your second tab should now look as follows:



With each step of this example, we have gotten more and more complex. At some point, it is very likely that an empty screen has greeted you instead of your application. Before we go too far ahead, we need to take a look at what you should do when things go boom.

## Testing and debugging the application

The first place to start when testing an application in Safari is the Error Console. From the **Develop** menu, select **Show Error Console**.



### Parse errors

The Error Console in the previous screenshot tells us two very important things. The first, is that we have **SyntaxError: Parse error**. This means that somewhere in the code we did something that the browser didn't understand. Typically, this is something such as:

- Forgetting to close a parenthesis, bracket, or brace, or adding an extra one
- Not having a comma between the configuration options, or adding an extra comma
- Leaving out the semicolon at the end of one of the variable declarations
- Not closing quotes or double-quotes (also not escaping quotes where necessary)

The second critical bit of information is `/app/TouchStart-4.js: 39`. It tells us that:

- `/app/TouchStart-4.js` is the file where the error occurred
- `39` is the line where the error occurred

Using this information, we should be able to track down the error quickly and fix it.

## Case sensitivity

JavaScript is a case-sensitive language. This means that if you type `xtype: 'Panel'`, you will get the following in the Error Console:

**Attempting to create a component with an xtype that has not been registered:  
Panel**

This is because Sencha Touch is expecting `panel` and not `Panel`.

## Missing files

Another common problem is missing files. If you don't point your `index.html` file at your `sencha-touch-debug.js` file correctly, you will get two separate errors:

1. **Failed to load resource: the server responded with a status of 404 (Not Found)**
2. **ReferenceError: Can't find variable: Ext**

The first error is the critical bit of information; the browser could not find one of the files you tried to include. The second error is caused by the missing file and simply complains that the `Ext` variable cannot be found. In this case, it's because the missing file is `sencha-touch-debug.js`, which sets up the `Ext` variable in the first place.

## Web Inspector console

Another feature of Safari Web Inspector that is incredibly useful for debugging applications is the console. In your JavaScript code, add the following command:

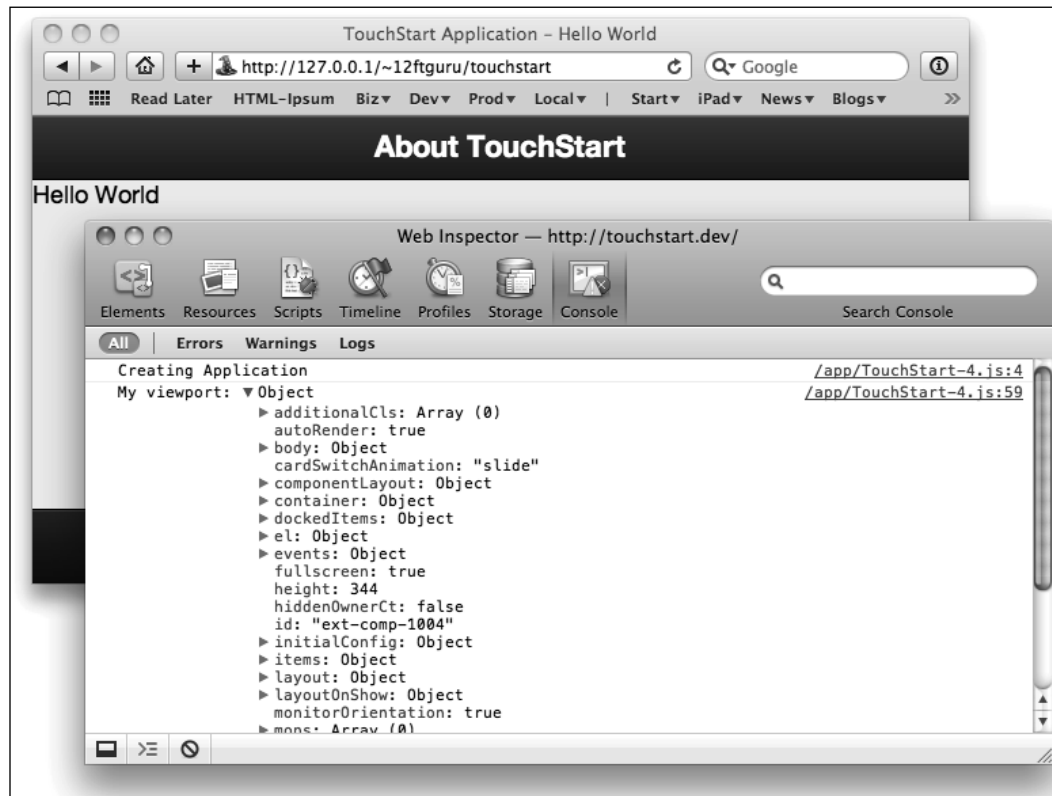
```
console.log('Creating Application');
```

Add it just before this new Application line:

```
new Ext.Application({
```

You should see the text **Creating Application** in your Web Inspector's console tab. You can also send variables to the console where you can view their contents, thus:

```
console.log('My viewport: %o', this.viewport);
```



This shows you the `TabPanel` component we created, if you place it after the `this.viewport = new TabPanel` block of code. This is useful if, for some reason, you have a component that is not displaying properly. Sending an object to the console allows you to see the object as JavaScript sees it.



If you'd like to learn more about using the Safari Web Inspector for debugging your application, visit Apple's *Debugging your Website* page at [http://developer.apple.com/library/safari/#documentation/AppleApplications/Conceptual/Safari\\_Developer\\_Guide/DebuggingYourWebsite/DebuggingYourWebsite.html](http://developer.apple.com/library/safari/#documentation/AppleApplications/Conceptual/Safari_Developer_Guide/DebuggingYourWebsite/DebuggingYourWebsite.html).

## Updating the application for production

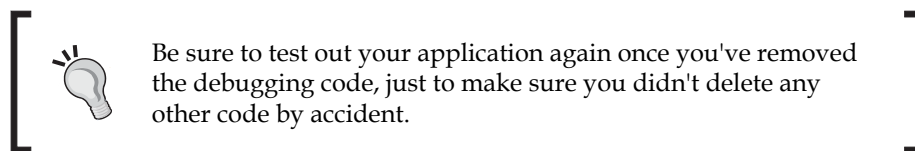
When you're done writing and testing your application, and are comfortable that it's ready for production, there are a few simple steps you should take before you release your application into the wild.

### Point to production library files

In our HTML file, we suggested loading the file `sencha-touch-debug.js` via the script tag. You should definitely change this to `sencha-touch.js`, in order to reduce load times and memory use. You may also have used the `sencha-touch-debug.css` file to help with writing your own custom CSS. This should be changed back to `sencha-touch.css` as well.

### Remove debugging code

You should also go through your application's JavaScript code and remove any `console.log` lines, alerts, or any other code you added to help you debug errors. Many mobile devices don't understand debugging code, and those that do may behave strangely if you leave it in place.



### Going that extra mile

There are some optional steps you could take before putting your application into production. With an application as simple as this one, these additional steps aren't really necessary. With larger applications, however, they can help to speed up your application and reduce its download size:

1. Minimize your JavaScript and CSS via a tool, such as YUI Compressor (<http://developer.yahoo.com/yui/compressor/>) or Google's Minify (<http://code.google.com/p/minify/>), to reduce file size.
2. Combine your separate graphics files into sprites to reduce load time.



The following are tools that can help you create sprites from your image files:

- SpriteMe - <http://sprite.me.org/>
- CSS Sprite Creator - <http://www.flowerind.com/sprite-creator/>

3. Prepare your application icon: create a 72px x 72px PNG file. Add the following to the <head> element of your index.html file: `<link rel="apple-touch-icon" href="icon.png"/>`.



Apple has a document explaining the guidelines for creating icons for your application, at <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/IconsImages/IconsImages.html>.

## Putting the application into production

Now that you've written and tested your application, and prepared it for production, it's time to find somewhere for it to live. Since the method for putting an application into production will vary based on your setup, we will be covering this task in very general terms.

The first thing to do is to familiarize yourself with three basic pieces of the puzzle for putting your application into production: **web hosting**, **file transfer**, and **folder structure**.

While it is fine to develop your application on a local web server, if you want anyone else to see it, you will need a publically accessible web server with a constant connection to the Internet. There are a large number of web hosting providers, such as *GoDaddy*, *HostGator*, *Blue Host*, *HostMonster*, and *RackSpace*.

Since our application is pure HTML/JavaScript/CSS, you don't need any fancy add-ons, such as, databases or server side programming languages (PHP or Java), for your web hosting account. Any account that can serve up HTML pages is good enough. The key to this decision should be customer support. Make sure to check the reviews before choosing a provider.

The hosting provider will also supply information on setting up your domain and uploading your files to the web server. Be sure to keep good track of your username and password, for future reference.

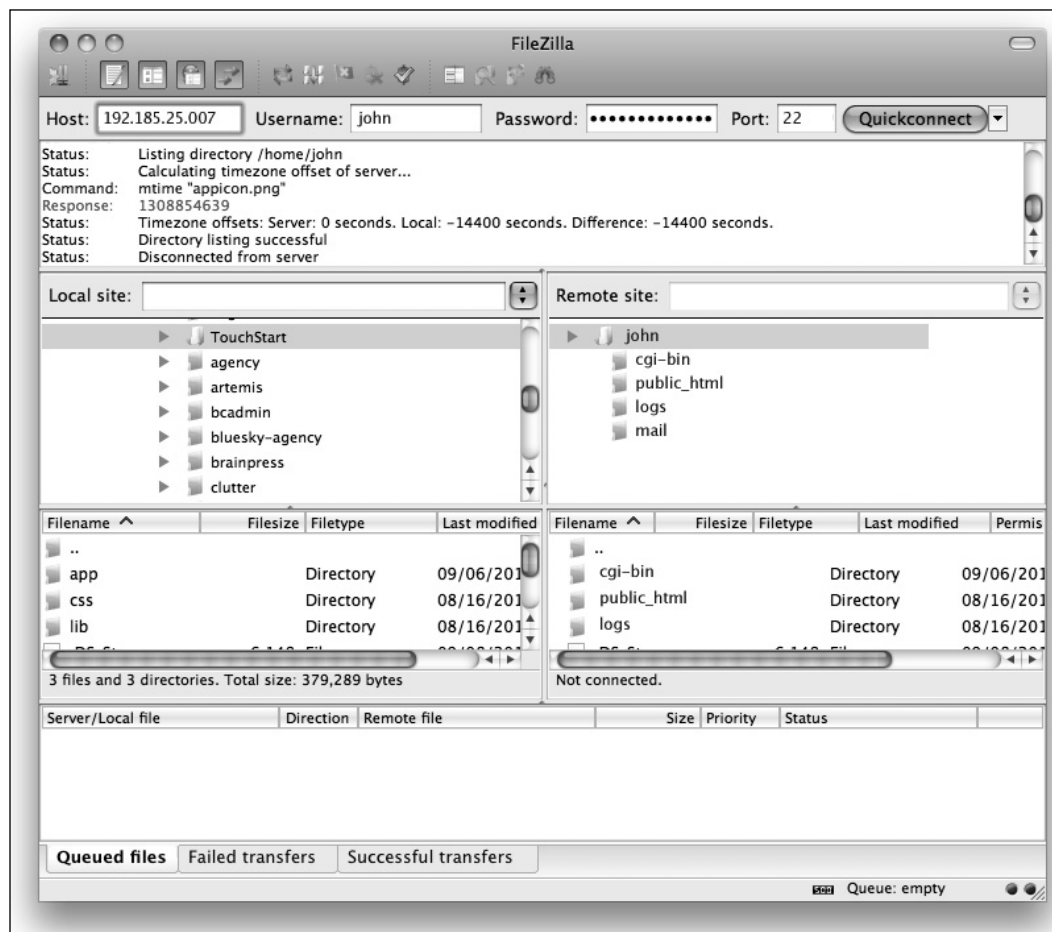


In order to copy your application to your web hosting account, you'll probably have to familiarize yourself with a **FTP (File Transfer Protocol)** program such as **FileZilla**. As with hosting providers, there is a huge selection of FTP programs. Most of them follow a few basic conventions.

To begin with, you will need to connect to the web server with the FTP program. You will need:

- A name or IP address for the web server
- Your web hosting username and password
- A connection port for the web server

Your web hosting provider should provide you with this information when you sign up.



Once you are connected to the server, you will see a list of files on your local machine, and files on your remote web server. You will need to move the **TouchStart** files on the remote server to upload them. Your hosting provider will also provide you with the name of a specific folder where these files need to go. The folder is typically called something like **httpd**, **htdocs**, **html**, or **public\_html**.

This brings us to our last consideration for uploading files: folder path.

The folder path affects how the application locates its files and resources. When you upload the application to the remote web server, it can affect how your folder is seen within the application. If you have any files referenced from an absolute path, such as `http://127.0.0.1/~12ftguru/TouchStart/myfile.js`, then the file will not work when you move things over to the web server.

Even relative URLs can become problematic when you transfer files to the remote server. For example, if you have a file which uses the path `/TouchStart/myFile.js`, and you upload the contents of the **TouchStart** folder instead of the folder itself, the file path will be incorrect.

This is something to keep in mind if you find yourself with missing images or other errors.

Again, your web hosting provider is your best resource for information. Be sure to look for *Getting Started* documentation and don't be afraid to seek help from any user forums that your hosting provider may have.

## Summary

In this chapter, we created our first simple application. We showed some of the basics of Sencha Touch components, including configuration and nesting of components within one another. We discussed the differences between `panel` and `container` components, and when to prefer one over the other; we also introduced you to the `TabPanel` and `list` components. In addition, we explained some basic debugging methodology and prepared our application for production.

In the next chapter, we will create a custom theme for our application through the use of SASS and the Sencha Touch library's styling tools.

## Where to buy this book

You can buy Sencha Touch Mobile JavaScript Framework from the Packt Publishing website: <http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book>.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



[www.PacktPub.com](http://www.PacktPub.com)

**For More Information:**

[www.packtpub.com/sencha-touch-mobile-javascript-framework/book](http://www.packtpub.com/sencha-touch-mobile-javascript-framework/book)