

## FAST TRACK TO SENCHA TOUCH

(This page intentionally left blank)

---

---

# **Unit 1:**

## **Unit 1: Introducing the Course**

### **About the Course**

Fast Track to Sencha Touch is designed to teach experienced web developers and designers how to design and implement web applications for the iPhone, iPad, and Android devices. The course is task-based, with students learning by doing. Over the three days, you will create and update a web application designed for citizens of the District of Columbia to research and report crimes in their neighborhoods.

Along with covering the basics of Sencha Touch, the course focuses on best practices and design, stressing the importance of usability, optimization, and maintainability of cross-device compatible applications.



*Illustration 1: Splash screens for the two applications you will build during class*

## Reviewing the Course Objectives

After completing this course, you should be able to:

- Develop applications for multiple form factors
- Implement effective layouts for your application
- Create data entry forms that submit to an application server
- Request JSON and JSON-P data from an application server
- Output hierarchically structured information using data drill-down
- Integrate Google Maps, geolocation, and charting into your applications
- Deploy audio and video
- Theme your application
- Optimize your application for deployment
- Compile to a native Android and iPhone app

## Reviewing the Course Prerequisites

The knowledge prerequisites for this course are:

- Prior experience with HTML 5
- A casual understanding of CSS
- Intermediate JavaScript coding skills and, in particular, familiarity with JavaScript Object Notation

## Required Software

The following software is REQUIRED to be installed on each student workstation:

- **Apache HTTP Web Server**  
<http://httpd.apache.org/>
- **Safari** (<http://www.apple.com/safari/>)
- **Aptana Studio 2** (<http://www.aptana.com>)
- **Sencha Touch**
- **Sencha Touch Charts**
- **Fast Track to Sencha Touch student files**  
<http://www.senchatraining.com/ftst.zip>

## Optional Software

The following software may be optionally installed on student workstations:

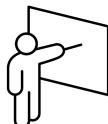
- **Android SDK** (<http://developer.android.com/sdk/index.html>)
- **PhoneGap SDK** (<http://www.phonegap.com>)
- **JSBuilder** (<http://code.google.com/p/js-builder/>)
- **MobiOne** (<http://www.genuitec.com/mobile/>)
- **ibdemo2 Cross Platform iPhone and iPad Web Browser Simulator**  
<http://code.google.com/p/ibbdemo2/>
- **iOS SDK**  
<http://developer.apple.com/devcenter/ios/index.action>
- **Adobe ColdFusion 9**  
<https://www.adobe.com/cfusion/tdrc/index.cfm?product=coldfusion>
- **MySQL Community Server 5.1+**  
<http://dev.mysql.com/downloads>

Complete installation instructions are covered in Appendix A of this guide.

## Reviewing the Course Format

This course is divided into ten units, each of which presents new information and contains demonstrations, walkthroughs, and a lab. At the end of each unit, you will find a summary and a short review to test your knowledge of the unit's content.

The following icons are used throughout the guide:



**Concepts** introduce new information.



**Demonstrations** illustrate new concepts.



**Walkthroughs** guide you, with the instructor's assistance, through procedures in a hands-on context.



**Labs** let you practice new skills on your own.



**Summaries** provide a brief synopsis of the unit's content.



**Reviews** test how well you remember the concepts from the unit.



**Best Practices** provide you with helpful insights and information.

# Outlining the Course Content

- **Unit 1: Introducing the Course**
- **Unit 2: Getting Started with Sencha Touch**
  - Introducing Sencha Touch
  - Setting up the development environment
  - Developing your first Sencha Touch application
- **Unit 3: Designing a Layout**
  - Using Panels
  - Using Toolbars, Icons, and Buttons
  - Using Box Layouts
  - Using the Card layout
  - Deploying Tabs
- **Unit 4: Working with Forms**
  - Using the FormPanel
  - Handling Text
  - Grouping content with Field Sets
  - Using checkboxes and radio buttons
  - Working with sliders, spinners, and toggles
  - Specifying a URL for posting, submitting, or loading
- **Unit 5: Making Data Requests**
  - Defining a connection to a server
  - Working with JSON / JSON-P
  - Defining a custom data model
- **Unit 6: Advanced GUI techniques**
  - Displaying Audio and Video
  - Integrating Google Maps
  - Visualizing Data with Sencha Touch Charts
- **Unit 7: Theming your Applications**
  - Using Compass and SASS
  - Theming your Sencha Touch Application
- **Unit 8: Optimizing Your Applications**
  - Caching your Application files on the Device
  - Caching Data on the Device
  - Applying Object-Oriented Techniques
  - Preparing your Application for Production

## Demonstration 1-1: Viewing the Applications



Observe as your instructor introduces the applications that you will be building during the walkthroughs and labs of this course.



*Illustration 2: The mobile phone splash pages for your walkthroughs and labs*

### Reviewing the CrimeFinder Application

You will build the CrimeFinder DC mobile application during your instructor-led walkthroughs. You will build the application in the following order:

- Confess to a Crime form
- Crime Listing
- FBI RSS Missing Children News Feed
- FBI Most Wanted News Feeds
- Video of Traffic Accident
- FBI Podcast
- Google Map displaying crimes

## Confess to a Crime Form

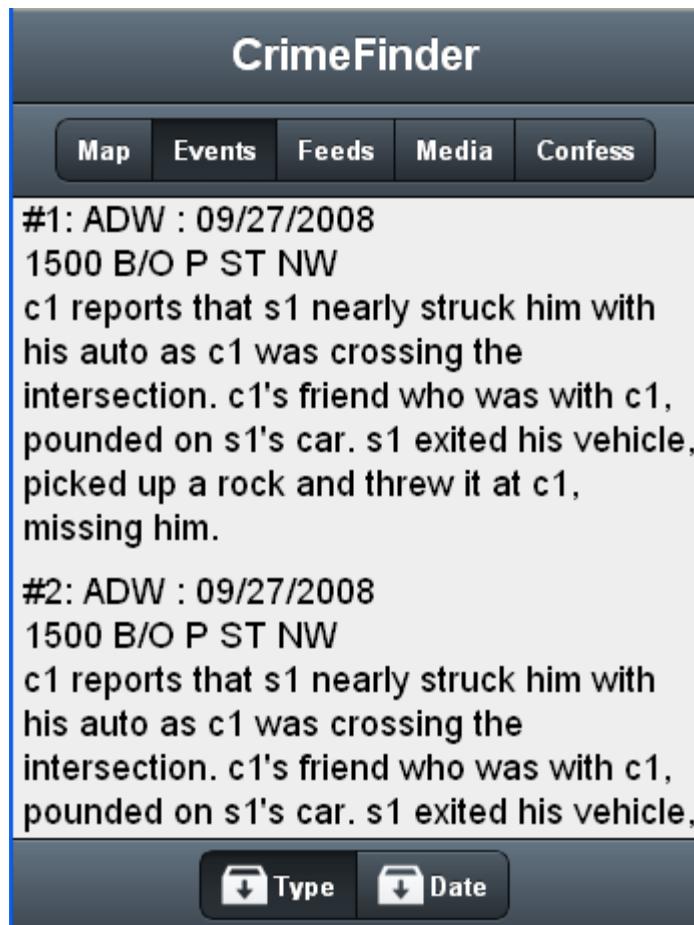
The Confess feature enables you to report a crime and be judged. The form features data validation, submits to an application server, and uses several HTML 5-specific form elements.

The screenshot shows a web application titled "CrimeFinder". At the top, there is a navigation bar with five tabs: "Map", "Events", "Feeds", "Media", and "Confess". The "Confess" tab is highlighted. Below the navigation bar, the main content area has two sections: "About me" and "Offense". The "About me" section contains three input fields: "Last Name", "First Name", and "E-mail". The "Offense" section contains two dropdown menus: one for "Jaywalking" and another for "Date". At the bottom of the form is a large button labeled "Prepare to be Judged".

*Illustration 3: Pay a fine and clear your conscience!*

## **Listing Crimes**

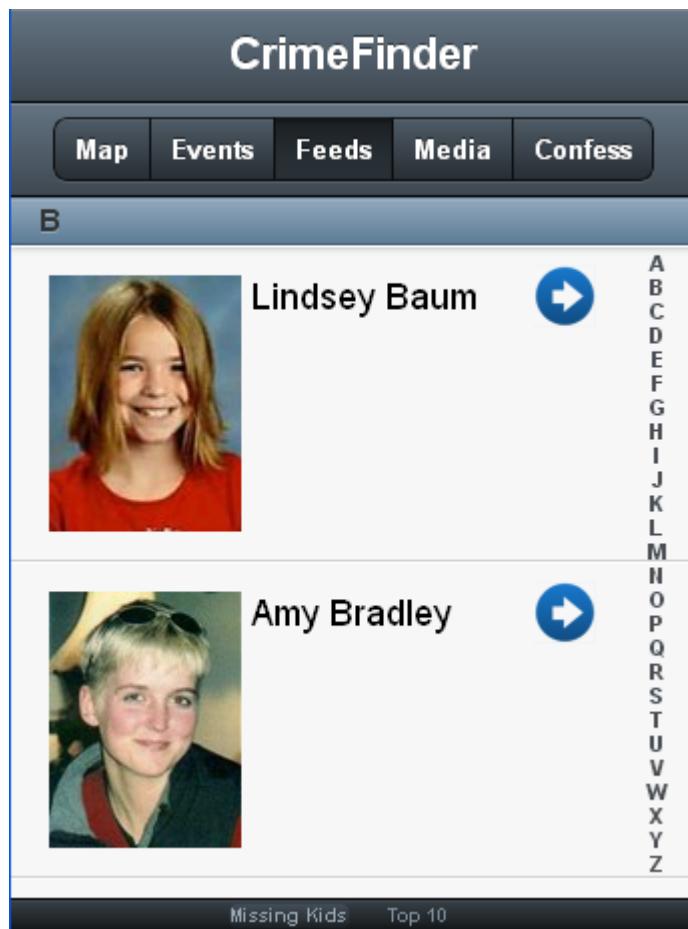
The crime listing feature pulls data from an application server via AJAX into a data model and displays it within a panel using a template. You have complete control of the output and will learn how to re-sort and automatically refresh information being displayed.



*Illustration 4: The crime listing page. It might be time to change neighborhoods!*

## FBI Missing Children

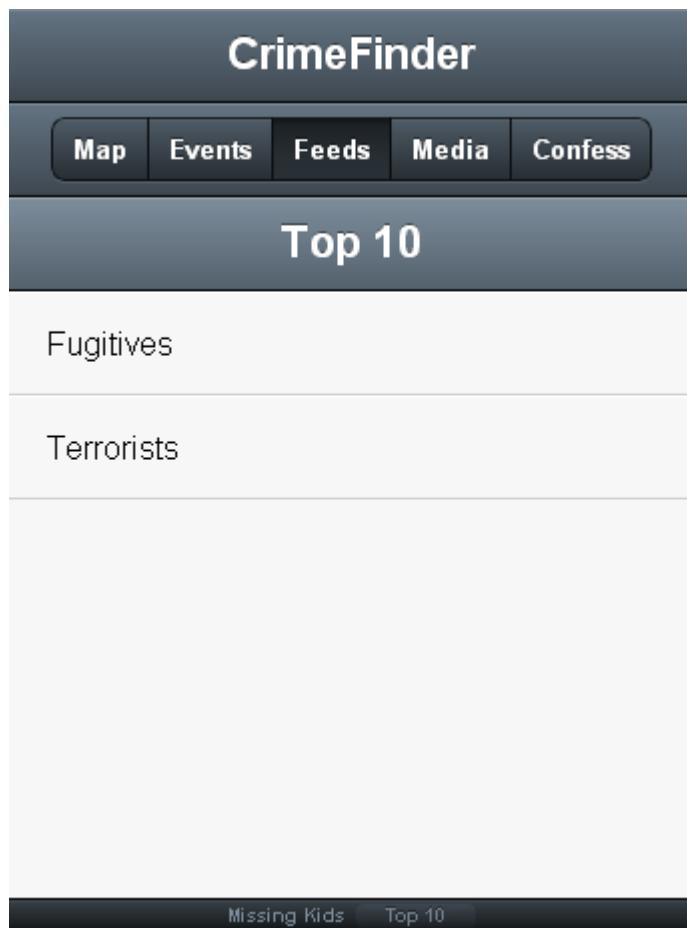
You will fetch the FBI missing children RSS feed using AJAX and display it using a Sencha Touch list component. Pressing the blue arrow button displays detailed information about the case, a common technique used data drill-down scenarios.



*Illustration 5: Have you seen these children?*

## FBI Most Wanted

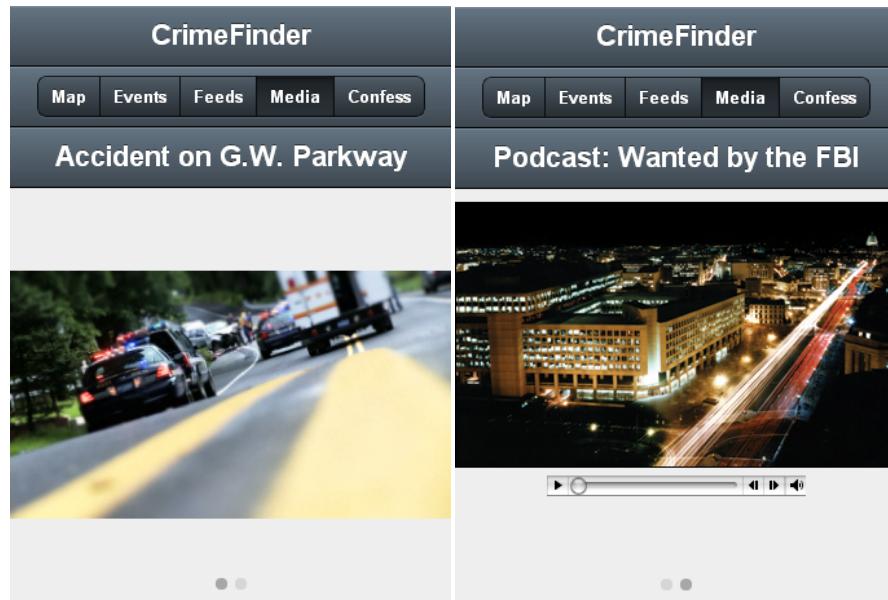
You will fetch the FBI 10 fugitive and terrorist news feeds using AJAX and display it using a Sencha Touch nested list component. Nested lists are a method of rendering hierarchical, tree-based data.



*Illustration 6: Bad Boys...Bad Boys...Watcha gonna do?*

## Working with Audio and Video

You will display a video of a traffic accident and enable users to play an MP3 podcast of an ongoing FBI investigation..

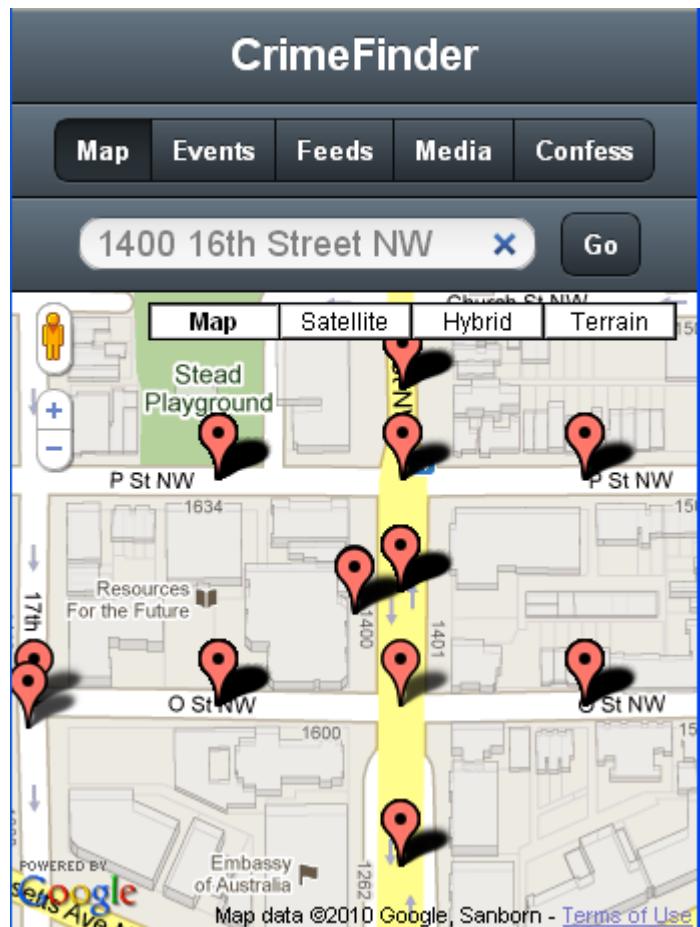


*Illustration 7: Video playback*

*Illustration 8: MP3 playback*

## Working with Google Maps

During this course you will learn how to display a Google map and display markers based on information fetched from a web service via AJAX..



*Illustration 9: Rough neighborhood? The Event listing shows the crime details.*

## Using Charts

You will learn how to use Sencha Touch Charts in order to visualize and analyze the relationships between data points.

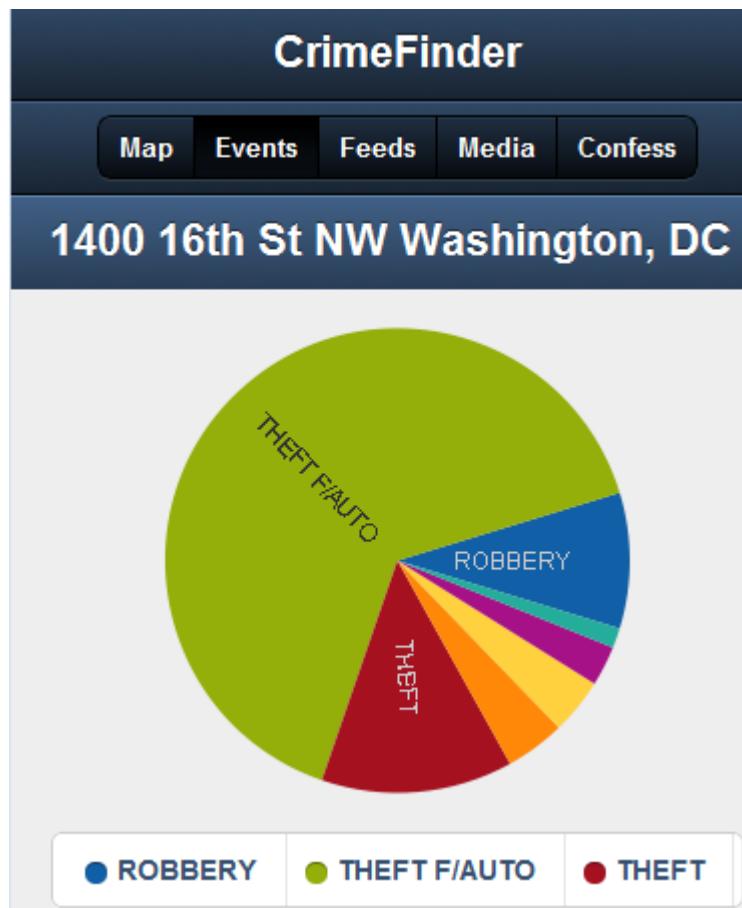
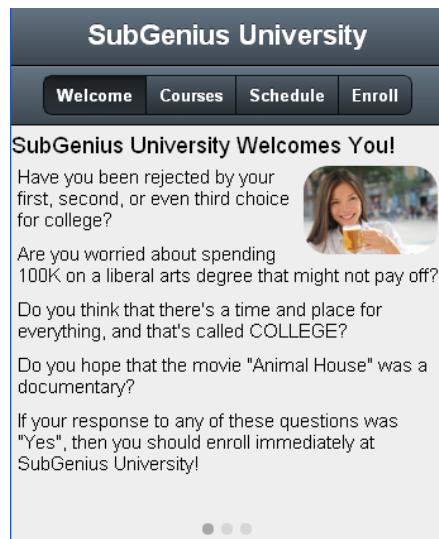


Illustration 10: The Events Chart View

## Reviewing the SubGenius University Web Site

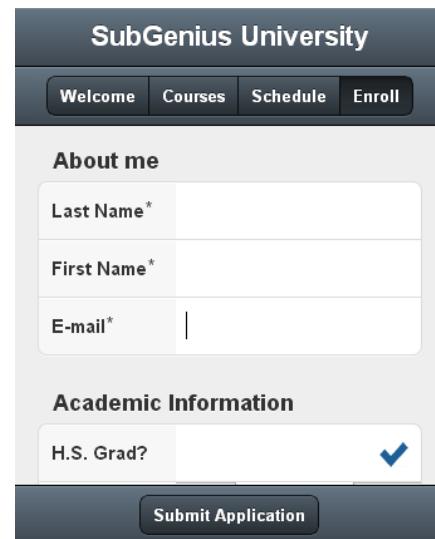
You will build the SubGenius University web site during your lab exercises. The application consists of the following panels:

- A Welcome Page containing a marketing pitch for the university
- A list of professors fetched from an AJAX request
- A Google Map, displaying campus locations
- A list of courses, fetched from an AJAX request
- A schedule of upcoming classes, fetched from an AJAX request
- An online enrollment form



The screenshot shows the SubGenius University Welcome page. At the top, there's a navigation bar with tabs: Welcome (which is active), Courses, Schedule, and Enroll. Below the navigation bar, the main content area has a heading "SubGenius University Welcomes You!". It includes a question about college rejection, a photo of a smiling person holding a mug, and several other questions related to college costs, time management, and movie preferences. At the bottom of the content area, there are three small circular dots.

*Illustration 11: The marketing pitch*



The screenshot shows the SubGenius University Enrollment form. At the top, there's a navigation bar with tabs: Welcome, Courses, Schedule, and Enroll. Below the navigation bar, the main content area has a section titled "About me". It contains three input fields: "Last Name\*", "First Name\*", and "E-mail\*". Below this, there's a section titled "Academic Information" with a dropdown menu showing "H.S. Grad?" followed by a checked checkmark. At the bottom right of the form is a large blue button labeled "Submit Application".

*Illustration 12: Enrollment form*

## Unit Summary



- The course is presented through a combination of lectures, demonstrations, walkthroughs, and labs.
- The course has been designed assuming that you already understand JavaScript, Javascript Object Notation, and HTML 5.
- The course consists of 8 units.
- The course focuses on developing applications for the iOS, and Android devices.
- You will build a web application to analyze neighborhood crime in Washington, DC.
- You will also build a mobile web site for the fictional SubGenius University
- This course teaches the fundamentals of developing applications for devices with Sencha Touch.

(This page intentionally left blank)

---

---

## **Unit 2: Getting Started with Sencha Touch**

### **Unit Objectives**

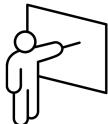
After completing this unit, you should be able to:

- Understand some of the design challenges that come with building mobile applications
- Understand the features and capabilities of Sencha Touch.
- Set up your development environment
- Create a simple “Hello World” application

### **Unit Topics**

- Understanding Design Challenges with Devices
- Introducing Sencha Touch
- Setting Up Your Development Environment
- Creating a “Hello World” Application

# Understanding Design Challenges with Devices



Supporting variable screen resolutions, touch input, gesture support, and variable bandwidth inevitably leads you to build applications that look very different from their desktop counterparts.

Consider the following table that illustrates the differences between some of the most popular mobile platforms:

Device	Resolution (pixels)	Screen Size (diagonal)
iPhone 3-series	320 x 480	3.5"
iPhone 4	640 x 960	3.5"
iPad	1024 x 768	9.7"
HTC Incredible (Android)	480 x 800	3.7"
Droid X	480 x 854	4.3"
Samsung Galaxy Tab	1024 x 600	7"

Such a significant variance between device capabilities and, in particular, the very small screens of mobile phones necessitate that you implement new user interface paradigms. One challenge, for instance, is that the iPhone Human Interface Guidelines from Apple recommend that all of your on-screen buttons should be a minimum of 42 pixels in height. This occupies nearly 10% of the screen on an iPhone 3-series! Based on that recommendation, using desktop development paradigms (which tend to be very button and menu-centric), you would ultimately produce an application that reserved a very small portion of real-estate to display the information that you wish to convey.

## Developing for Phones

The Apple iPhone Human Interface Guidelines outline a number of best practices for mobile development including the following basic principles:

- Do not display UI elements that are not absolutely necessary. Minimize the number of user controls and required input elements.
- Since users can only see one screen at a time, you will likely need to reorganize how information is presented from your desktop application.

- Using your application needs to be completely obvious. Use standard controls for which users have had prior experience. Provide UI elements that enable a user to go back and retrace their steps.
- Users typically will use your application for very brief periods of time, and usually under distracting conditions. Streamline your processes to enable users to complete them as quickly as possible.
- Minimize text input. Use table views, sliders, select boxes, etc.
- Limit gestures to “tap & drag” unless absolutely necessary.
- All buttons and “clickable” areas should be fingertip-sized (44x44)
- For optimized web page-based applications, create a custom icon that users can put on their Home screen
- Avoid technical jargon
- Branding should be subtle and understated. For instance, do not display a “splash screen” that would delay a user from using your application.
- Launch in portrait orientation, however, your application should support both orientations.
- Restore state to display the identical screen from when the user last exited your application. Conversely, you should save the application state early and often.
- On the iPhone, make sure your UI can adjust to the double-high status bar that appears during in-progress phone calls.
- In-application advertisements should be sized as 50 pixels in height in portrait mode and 32 pixels high in landscape. Ads should be placed at the bottom of the screen.
- Use the user's location whenever possible.
- There are three types of applications
  - Productivity applications (such as the mobile crime tracker) that organizes information hierarchically using multiple views
  - Utility applications that perform a single task (such as viewing current weather conditions) and employ a simple, attractive GUI
  - Immersive applications, such as games or simulations, that present the user with a custom GUI.

*Note: This class focuses on developing productivity and utility applications with Sencha Touch.*

You can download the entire iPhone HIG at the following url:

<http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf>

## Developing for Tablets

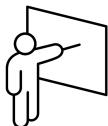
As indicated by the preceding table, tablets have a screen resolution that more closely approximates a desktop experience. Some devices, such as the iPad, have a keyboard dock available. When developing for tablets, consider the following:

- In all likelihood you will need to redesign your application to take full advantage of the increased screen resolution. For instance, with the new form factor you can probably reduce the number of screens required for data-drill down. You may be able to combine two or more screens into a single view. This should also enable you to reduce full-screen transitions.
- Redesigning an application for a tablet should not alter its core feature set. Control structures, such as toolbars, should remain largely unchanged.
- With the added screen area you have more flexibility in implementing your branding strategy and “beautifying” your application. For instance, you can use photo-realistic art work to mimic real-world objects. Remember to stay focused on your content presentation, however, as the best navigation remains unobtrusive.
- You should avoid radical changes to your page layout when there's a shift in orientation. With more screen space available, you should be able to provide a consistent UI regardless of whether you are in portrait or landscape mode.
- The added screen real-estate also enables you to innovate by designing applications for two people using the device at the same time (i.e. an interactive chess match).
- Pad-based applications should minimize modality – allow users to interact with your application in non-linear ways
- The larger screen opens up more possibilities for using gestures.
- Move all toolbars to the top of the screen.

The Apple iPad Human Interface Guidelines are published at the following url:

<http://developer.apple.com/library/ios/documentation/General/Conceptual/iPadHIG/iPadHIG.pdf>

# Introducing Sencha Touch



Sencha Touch is the world's first app framework built specifically to leverage HTML5, CSS3, and Javascript for the highest level of power, flexibility, and optimization.

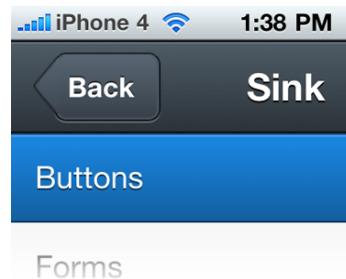
As discussed in the previous section, the design issues that you are facing are not inconsequential. Not only is there a wide disparity between device screen resolutions, but there are also differences between the capabilities of the web browsers that each device supports. The only way to write a single application that is compatible across the widest variety of devices is to use a framework – and Sencha Touch is the best one available.

Besides mitigating the problems associated with disparate screen resolutions, Sencha Touch also enables you to easily theme your application using CSS, provides a comprehensive set of icons for use in toolbars tabs, and supports a robust animation system – including 3D transformations like flip and cube.

It's built-in layouts mimic standard UI widgets on iOS and Android, enabling you to build cross-platform compatible applications using familiar toolsets - JavaScript, HTML 5, and CSS.

Functionality in Sencha Touch can be broken down into the following categories:

- User Interface
- Forms
- Animations
- Touch Events
- Data
- Media



**Illustration 1: Sencha Touch enables you to support varied resolutions**

## Designing GUIs in Sencha Touch

Sencha Touch supports a number of commonly used iOS layouts including, but not limited to, the following:

Type	Description
Carousel	Enables a user to slide left-right or up-down in order to bring different child items into view.
Overlay	Modal popups that can be attached to a panel or display centered on the screen
Tab Panel	A Container which can hold other components to be accessed in a tabbed interface Tab controls can be docked at either the top or the bottom of the screen.
List	A selectable list of items, typically paired with an alphabet index.
Toolbar	A set of docked buttons that are typically positioned at either the top or bottom of a panel. Sencha touch supports all the commonly used button and icon styles found in iOS and Android.
Map	Displays a map using the Google Maps API.

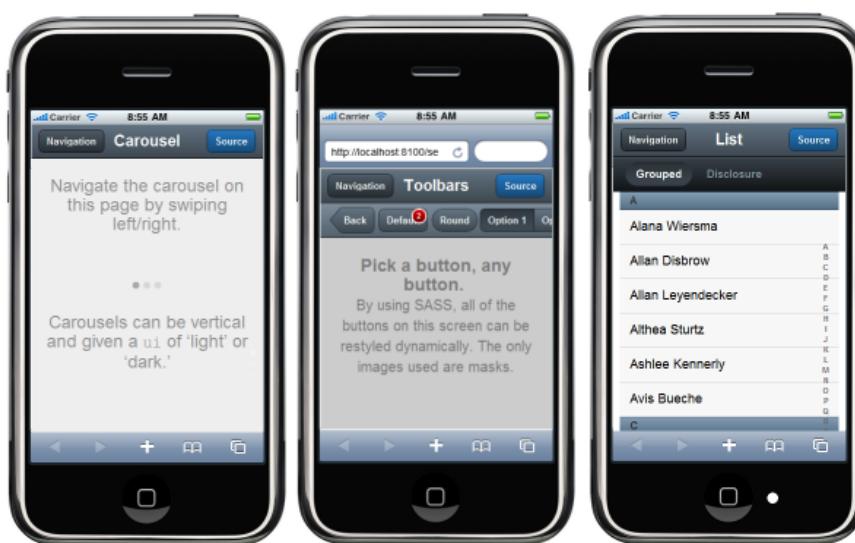
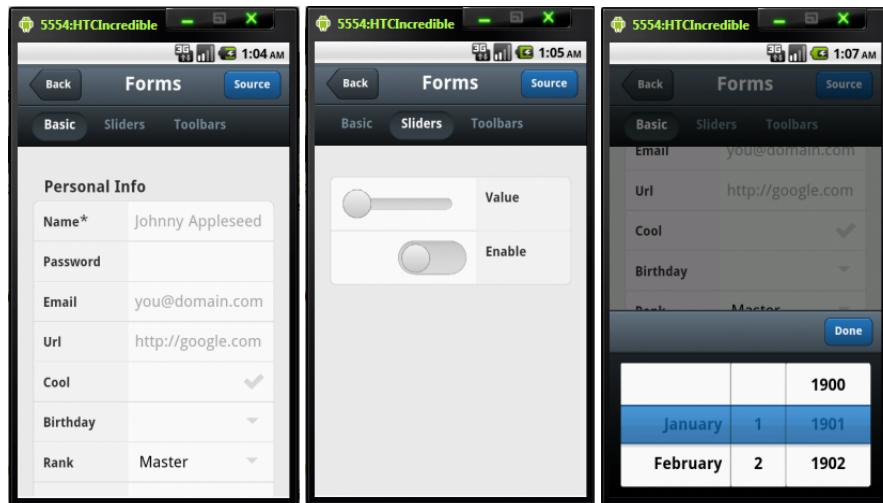


Illustration 2: Sencha supports Carousels, Toolbars, and Lists

## Working with Forms in Sencha Touch

Sencha touch supports all of the standard HTML 5 form 2.0 widgets including text fields (with validation and placeholders ), sliders, toolbar items, and selection list assistants as depicted below:



**Illustration 3:** Sencha Touch supports all commonly used form input widgets

## Using Animation to Illustrate Context Switching

Animation plays an important role with device-based web sites. by helping to show the user that there has been a transition in the application state.

Sencha Touch supports the following animations:

- Slide (cover)
- Pop
- Flip
- Slide (reveal)
- Fade
- Cube

## Handling Touch Events

Sencha Touch comes with a multitude of touch events. Typically, when a user interacts with a touchscreen, multiple events are triggered as part of a single interaction. The following events are supported:

- touchstart
- touchmove
- touchend
- touchdown
- scrollstart
- scroll
- scrollend
- tap
- tapstart
- tapcancel
- doubletap
- taphold
- swipe
- pinch
- pinchstart
- pinchend

## Handling Data

Sencha Touch provides methods for performing background data retrieval with AJAX. It contains a number of classes that enable you to work more efficiently with JavaScript Object Notation, enables you to define data models, interact with RESTful services, and – of course, work with XML.

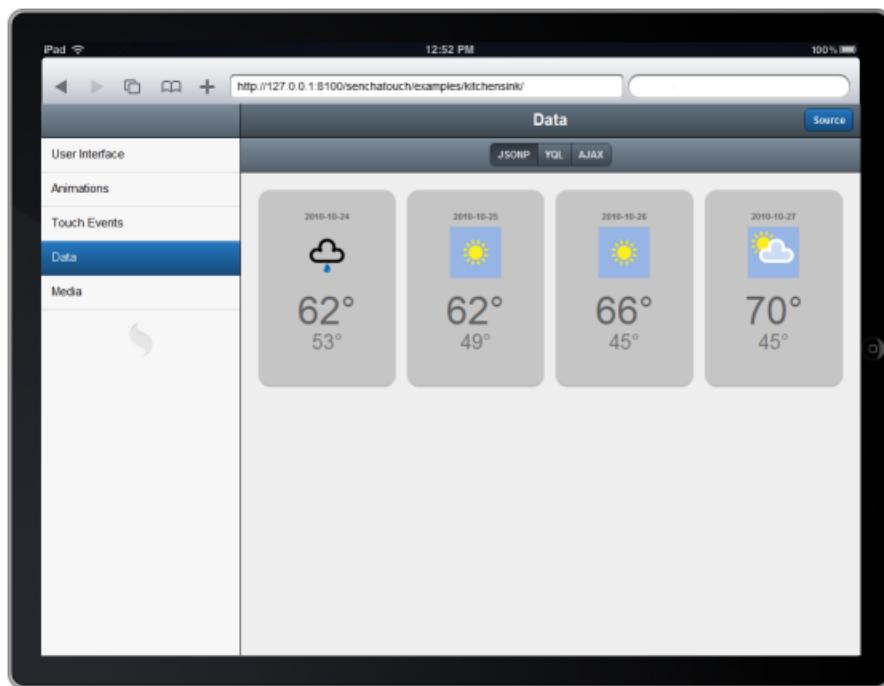


Illustration 4: Rendering JSON-P data visually on an iPad

Sencha Touch also interacts with YQL – the Yahoo Query Language which provides JSON APIs for traditionally REST-based services.

## Handling Media

As illustrated by the screen shot pictured at right, Sencha Touch can also be used to display HTML 5 audio and video. Videos can be linked with an image-based poster that displays prior to video playback. Recommended audio types include:

- MP3
- Uncompressed WAV and AIF
- AAC-LC or HE-AAC

Recommended HTML 5 video formats include:

- Mp4
- OGG



## Walkthrough 2-1: Introducing Sencha Touch



In this walkthrough, you will perform the following tasks:

- Review the Sencha Touch Documentation
- Review the Sencha Touch Examples

### Steps

#### Review the Sencha Touch Documentation

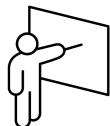
1. Open Safari and browse to <http://www.sencha.com>
2. Click on the Sencha Touch API Documentation link
3. Review the documentation with your instructor. Note the following:
  - The **Find a Class** box acts as a filter for the various Sencha Touch classes
  - You can perform a full text search from the **API Home** tab
  - Most Sencha Touch components inherit from other classes or components and were implemented using the Observer design pattern
  - You can filter the list of properties and methods for a specific component by clicking on the **Hide Inherited Members** button, located in the top right corner of the UI



#### Review the Sencha Touch Examples

4. Browse to <http://localhost:8100/senchatouch/>
5. Click on the **View Examples** button
6. Review the Kitchen Sink application with your instructor. Note that you can click on the **Source** button in any example to view its source code.

# Setting Up Your Development Environment



Sencha Touch comes bundled as a .ZIP file. You can simply deploy it to an HTTP-accessible directory on a web server and get right to work.

If you are running this courseware on a PC, we recommend installing the following applications to enhance your productivity:

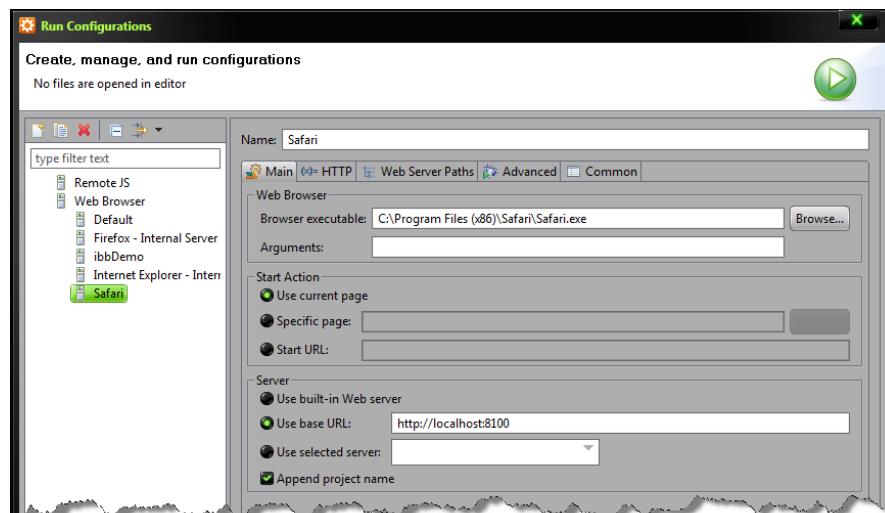
- Aptana Studio 2
- Apache Web Server
- Apple Safari (to act as an iPhone/iPad simulator)
- Android SDK (which contains an Android simulator - optional)

If you are developing on a Mac, in addition to the aforementioned software, we also recommend installing the Apple iPhone SDK as it has the best iPhone/iPad emulator available.

## Setting up a project in Aptana Studio

Aptana Studio, from Aptana Inc. is a free code editor with exceptionally robust support for HTML, CSS, and – in particular – JavaScript development. Furthermore, it directly supports Ext-JS, enabling you to easily transition between desktop and mobile development.

Setting up a project in Aptana Studio is a relatively straightforward affair. You should also create a debug configuration in Aptana that uses Safari as its browser as indicated by the following screen:



**Illustration 5: Create a run configuration that launches Safari**

## Testing with Simulators

Clearly there is no better test for your applications than actually executing them from your targeted devices. Note that if your device is on the same wireless network as your development machine, you should be able to access your development web server by its local ip address and port number – contingent on your machine's firewall settings.

Notwithstanding the above, the iterative and fast-paced nature of debugging and troubleshooting does not lend itself to testing with physical devices. Typically, developers of mobile applications use a variety of simulators to enhance their productivity while reserving testing on physical devices for final QA processes.

The following device simulators generally provide acceptable results for initial debugging:

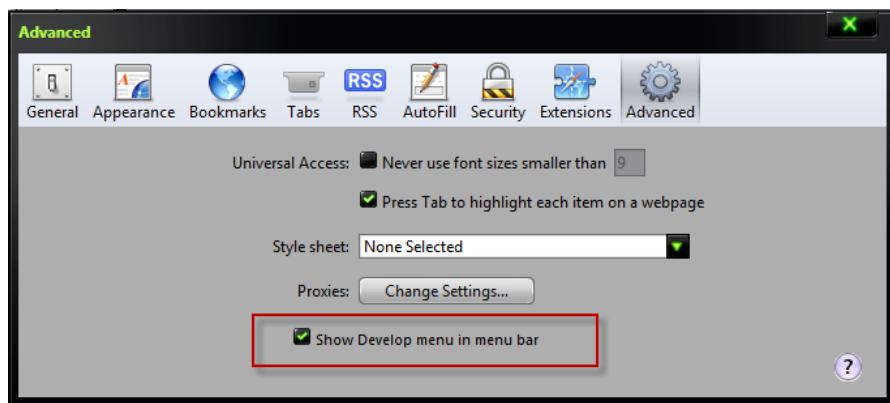
- Apple Safari (Windows) to simulate iOS
- Use 3<sup>rd</sup> Party Simulators (Windows) for iOS
- The Cross-Platform Android Simulator in the Android SDK
- The iPhone/iPad simulator that ships with the iOS SDK (Mac) from Apple



**Illustration 6: Simulators for Android and iOS**

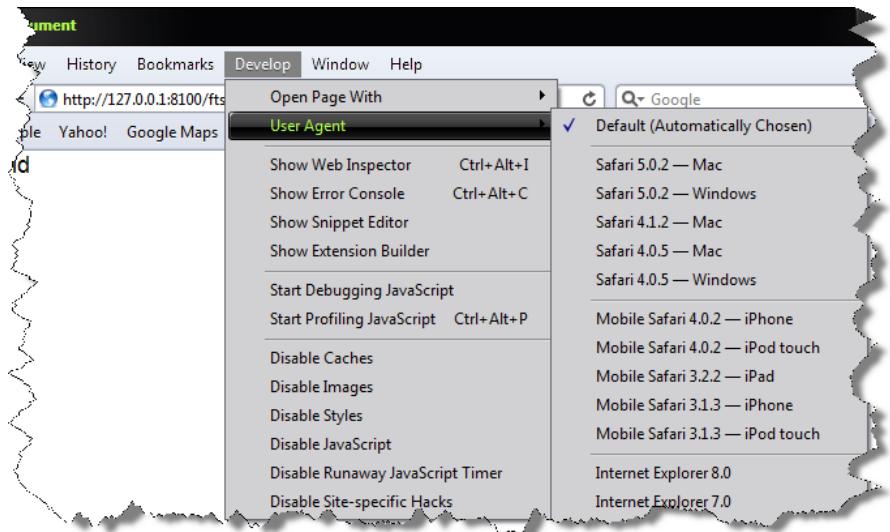
## Using Safari as an Apple Device Emulator

Sadly, Apple has chosen to not make their iOS SDK nor its device simulator available to PC users. As a result, the best way to test your application in development is to simply use the Safari browser. The first step in making Safari ready for development is to go into Preferences > Advanced and activate the Develop menu as indicated below:



**Illustration 7: Activating the Develop menu in Safari**

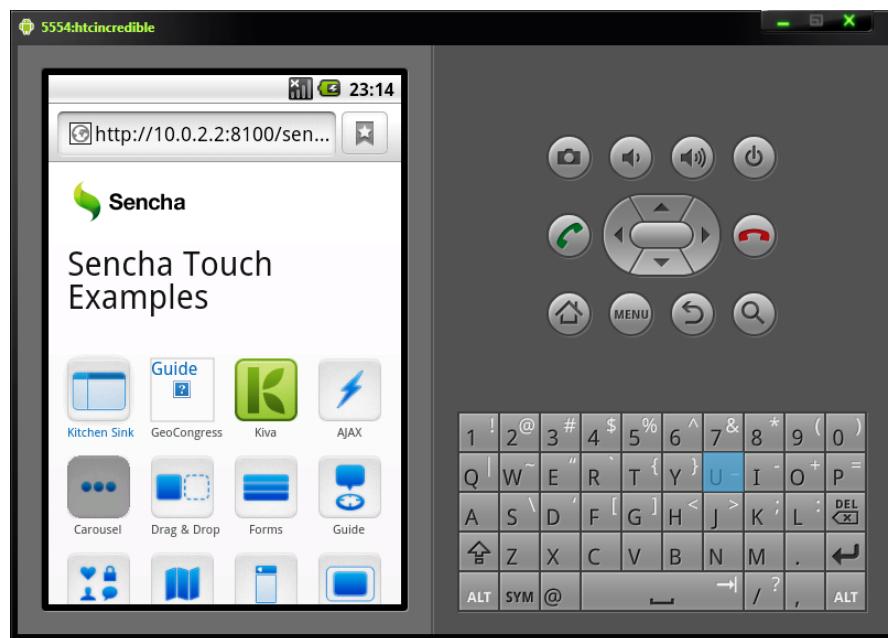
From the Develop menu, depicted below, you can choose the user agent that the browser will report back to web applications. You can also show the JavaScript error console and disable caching.



**Illustration 8: The Safari Develop menu allows you to set the User Agent**

## Testing with the Android Emulator

The Android emulator, depicted below, is a component of the Android SDK. It mimics all of the characteristics of a mobile device, however, it cannot place or receive calls.



**Illustration 9: The Android simulator can be configured for multiple form factors**

The emulator runs Android Virtual Device (AVD) configurations. Using the Android SDK you can configure each AVD for a specific Android OS, device screen size, and screen resolution. It will even simulate different bandwidth conditions.

Each instance runs behind a virtual router/firewall service that prevents it from accessing your development machine's network interfaces and settings and from the internet. An emulated device can not see your development machine or other emulator instances on the network. Instead, it sees only that it is connected through Ethernet to a router/firewall.

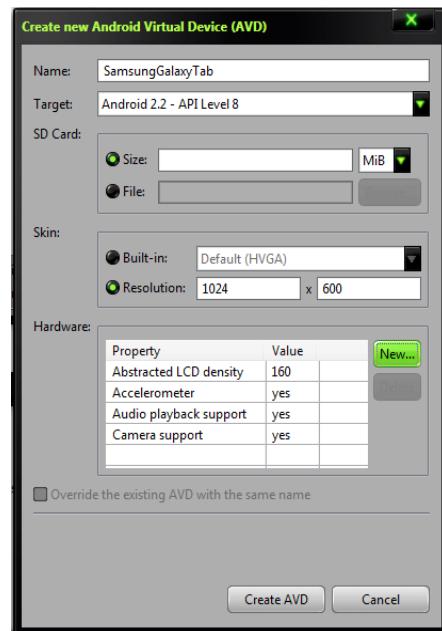
The virtual router for each instance manages the 10.0.2/24 network address space. The ip address 10.0.2.2 is a special alias that points back to 127.0.0.1 on your local development machine.

As depicted in the figure, you can create as many AVD's as you see fit in order to test the broadest range of devices. Each device can target a specific version of Android, an optional SD card, screen resolution, and additional hardware including, but not limited to the following:

- Dpad
- Accelerometer
- Camera
- Track Ball Support
- Touch screen
- GPS
- SD Card
- Keyboard

When the emulator is running, you can interact with the emulated device just as you would with an actual one except that you use your mouse pointer to "touch" the touchscreen and your keyboard keys to "press" the simulated device keys.

The following table summarizes key mappings between the emulator and your keyboard:



**Illustration 10: Configuring an Android Device**

Keyboard	Emulated Device Button
Home	Home
F2 / Page-Up	Menu
ESC	Back
F5	Search
Keypad 7	Portrait Mode
Keypad 9	Landscape Mode
F7	Power Off

Define and start AVD's through the /tools/android.bat application in your Android SDK.

## Using the iOS SDK Simulators

The best simulator for iOS comes, predictably, from Apple as part of their iOS SDK. In order to download the SDK you will need to register as an Apple developer and pay a \$99 fee to have the privilege of downloading the 3GB dmg file. Unpacked, the SDK occupies about 8GB of total disk storage.

For Sencha development, the only component of the SDK that you will use is the iOS simulator that enables you to toggle between iPhone 3, iPhone 4, and iPad modes. It is located in the following directory:

Mac HD: Developer: Platforms: iPhoneSimulator.platform: Developer: Applications

The following hotkey combinations affect the simulator.

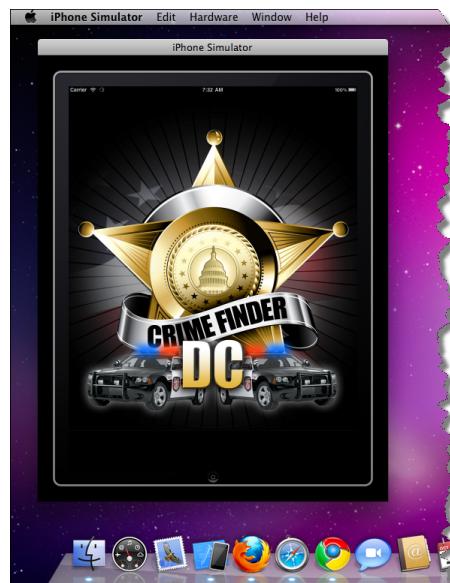


Illustration 11: Simulating an iPad in MacOS

Note that you can also toggle the in-call status bar and the device (iPad/iPhone/iPhone4) by clicking the option on the Hardware menu.

Key Combination	Description
Command + Left Arrow	Rotate Left
Command + Right Arrow	Rotate Right
Shift + Command + Z	Shake Gesture

## Walkthrough 2-2: Getting Started



In this walkthrough, you will perform the following tasks:

- Define Virtual Android Devices
- Configure Safari for Development
- Configure Aptana Studio for Development

### Steps

#### Configure Safari for Development

1. Click **Start > All Programs > Safari**
2. Select **Edit > Preferences > Advanced**
3. Check the checkbox labeled **Show Develop menu in menu bar**
4. Close the **Advanced** dialog box
5. Browse to [www.yahoo.com](http://www.yahoo.com)
6. Click on the **Develop** menu
7. Select **User Agent > Mobile Safari 4.02 iPhone**. Notice how the content layout changes.
8. Select **Develop > Disable Caches**.
9. Select **Develop > Show Error Console**.
10. Click on the **Scripts** tab and enable debugging
11. Review the debugging options with your instructor

#### Configure Aptana for Development

12. Open Aptana Studio by clicking **Start > All Programs > Aptana > Aptana Studio 2**
13. From the Aptana menu bar, select **Run > Run Configurations**
14. Click the button to add a new configuration and enter the following:
  - **Name:** Safari
  - **Browser Executable:** C:\Program Files(x86)\Safari\Safari.exe
  - **Start Action:** Use current page
  - **Server:** Use base URL: <http://127.0.0.1:8100/>
  - **Append Project Name:** checked
15. Click **Apply**

16. Click on the Common tab and turn on the checkboxes located in the **Display in favorites menu** block
17. Click **Apply**
18. Click **Close**
19. In the **File** view, right-click on **Projects** and select **New > Project**
20. Select **Web > Default Web Project** and click **Next**
21. Enter a project name of **ftst**
22. Turn off the checkbox labeled **Use default location**
23. Click the Browse button and select the **C:\apache\htdocs\ftst** directory
24. Click **OK**
25. Click **Next**
26. Click **Finish**
27. Review the directory structure of your class files with your instructor.

#### **Configure the Android Simulator (optional)**

28. Run **c:\android-sdk-windows\tools\android.bat**
29. Click on **Virtual Devices**
30. Click the **New** button
31. Enter the following information:
  - Name: **HTCIncredible**
  - Target: **Android 2.2 API Level 8**
  - Resolution: **480 X 800**
  - Abstracted LCD density: **240**
32. Click the **New...**button and add the following properties, one-at-a-time:
  - Accelerometer: **yes**
  - Audio playback support: **yes**
  - Camera Support: **yes**
  - GPS Support: **yes**
  - Touch-screen support: **yes**
  - Keyboard: **yes**
33. Click **Create AVD**

#### **Define a Virtual Android Tablet (optional)**

34. Click the **New** button

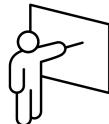
35. Enter the following information:
  - Name: **SamsungGalaxyTab**
  - Target: **Android 2.2 API Level 8**
  - Resolution: **1024 X 600**
  - Abstracted LCD density: **240**
36. Click the **New...**button and add the following properties, one-at-a-time:
  - Accelerometer: **yes**
  - Audio playback support: **yes**
  - Camera Support: **yes**
  - GPS Support: **yes**
  - Touch-screen support: **yes**
  - Keyboard: **yes**
37. Click **Create AVD**

#### **Start the Virtual Devices (optional)**

38. Click on the **HTCIncredible** AVD
39. Click Start and enter the following details:
  - Scale display to real size: **Checked**
  - Screen Size: **3.7**
40. Click **Launch**
41. Click on the **SamsungGalaxyTab** AVD
42. Click Start and enter the following details:
  - Scale display to real size: **Checked**
  - Screen Size: **7**
44. Click **Launch**

– End of Walkthrough --

## Creating a “Hello World” Application



Developing a Sencha Touch application starts with you creating an HTML 5 page that contains the following references:

- A `<link>` tag referencing the Sencha Touch CSS file
- A `<link>` to a custom CSS file referenced by your application
- A `<script>` tag that loads the Ext Touch library
- A `<script>` tag that loads your application's JavaScript file.

Therefore, your typical application would start with an HTML page resembling the following code:

```
<!DOCTYPE HTML>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
<title>Sencha Touch Template</title>

<!-- sencha touch css -->
<link rel="stylesheet"
href="../../senchatouch/resources/css-debug/sencha-
touch.css" type="text/css" />

<!-- application library -->
<link rel="stylesheet"
href="css/application.css"
type="text/css" />

<!-- sencha library -->
<script type="text/javascript"
src="../../senchatouch/sencha-touch-debug.js"></script>

<!-- application javascript -->
<script type="text/javascript"
src="scripts/index.js"></script>

</head>

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

*Note: Before putting the application into production, replace the `<script>` reference to `sencha-touch-debug.js` with a reference to `sencha-touch.js`*

## Initializing your Application with Ext.setup()

The very first call that you make within your application's javascript file is to Ext.setup() which takes a single javascript object as an argument. That JavaScript object supports the following properties:

Property	Description
fullscreen	Boolean. Outputs an appropriate meta tag for Apple devices to run in full-screen mode.
tabletStartupScreen	String. Startup screen to be used on an iPad. The image must be 768x1004 and in portrait orientation.
phoneStartupScreen	String. Startup screen to be used on an iPhone or iPod touch. The image must be 320x460 and in portrait orientation.
icon	String. Default icon to use. This will automatically apply to both tablets and phones. These should be 72x72
tabletIcon	String. An icon for only tablets. (This config supersedes icon.) These should be 72x72.
phoneIcon	String. An icon for only phones. (This config supersedes icon.) These should be 57x57.
glossOnIcon	Boolean/ Add gloss on icon on iPhone, iPad and iPod Touch
statusBarStyle	String. Sets the status bar style for fullscreen iPhone OS web apps. Valid options are <code>default</code> , <code>black</code> , or <code>black-translucent</code>
scope	Scope for the onReady configuration to be run in. Defaults to <code>this</code> .
onReady	JavaScript function that is executed once the DOM is ready.

Note that iOS will automatically add drop shadows, rounded corners, and gloss to your icons. Develop your icons with 90 degree corners, no gloss, and without alpha transparency. The images should be in PNG format.

The following example illustrates using Ext.setup():

```
Ext.setup({
    tabletStartupScreen: 'tablet_startup.png',
    phoneStartupScreen: 'phone_startup.png',
    icon: 'icon.png',
    glossOnIcon: false,
    onReady: function() {

    }
});
```

## Defining your Initial Layout with onReady

Inside the onReady() function is where you define the initial layout for your application. From here you can choose to deploy boxes, cards, panels, toolbars, tabs, and more. UI elements can be nested inside of other elements. For example, a panel could act as a parent for a toolbar and a list.

Panels are one of the basic layout building blocks. You can define a panel using the new Ext.Panel() constructor and output static html by passing in an html attribute as part of a JavaScript object as outlined below:

```
Ext.setup({
    tabletStartupScreen: 'tablet_startup.png',
    phoneStartupScreen: 'phone_startup.png',
    icon: 'icon.png',
    glossOnIcon: false,
    onReady: function() {
        var pnl = new Ext.Panel({
            fullscreen: true,
            html: 'Hello World'
        })
    }
});
```

Note: You will cover the Ext.Panel() method in more detail in unit 3

You can also load content from an HTML page using the contentEl attribute that references a container element referenced by ID as illustrated by the example on the following page:

```
<!DOCTYPE HTML>

<html>
<head>
<meta charset="utf-8">
<title>Alternate Hello World</title>

<!-- sencha touch css -->
<link rel="stylesheet"
      href="/senchatouch/resources/css-debug/sencha-touch.css"
      type="text/css" />

<!-- sencha library -->
<script type="text/javascript"
       src="/senchatouch/sencha-touch-debug.js"></script>

<!-- application javascript -->
<script type="text/javascript">
    Ext.setup({
        tabletStartupScreen: 'tablet_startup.png',
        phoneStartupScreen: 'phone_startup.png',
        icon: 'icon.png',
        glossOnIcon: false,
        onReady: function() {
            var pnl = new Ext.Panel({
                fullscreen: true,
                contentEl: 'myContent'
            })
        }
    })
</script>
</head>

<body>
    <div id="myContent">Hello World</div>
</body>
</html>
```

## Walkthrough 2-3: Initializing the CrimeFinder



During this walkthrough, you will develop the code necessary to start laying out the CrimeFinder application. You will add functionality to the CrimeFinder during the rest of the walkthroughs in this class.

- Create an HTML 5 home page for your application
- Invoke the Ext.Setup() method

If you are using the iPhone simulator on the mac, you will be able to add the html page to your home screen and view the startup page once it is pressed as indicated by the following screen shot:



Illustration 12: The iPhone simulator with CrimeFinder icon

## Steps

### Create the Application Home Page

1. Open **Aptana**
2. Right-click on the **walk/walk2-3** in the **ftst** project
3. Select **New > Other...**
4. Select **Web Files > HTML File** and click **Next**
5. Enter a file name of **index.htm** and click **Finish**
6. Change the document title to “**CrimeFinder**”
7. Change the **<!DOCTYPE>** declaration to indicate that the page is using HTML5
8. Update the **<meta>** tag to the HTML5 standard
9. Inside the **<head>** section of the document, insert a **<link>** tag that points to the Sencha Touch debug style sheet. Your code should appear similar to the following:

```
<link rel="stylesheet"
      href="/senchatouch/resources/css-debug/sencha-touch.css"
      type="text/css" />
```

10. After the code that you inserted from the previous step, insert a **<script>** tag to load the Sencha Touch debug API. Your code should appear similar to the following:

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

11. After the code that you inserted in the prior step, insert a **<script>** block.

12. Inside the **<script>** block, invoke the **Ext.setup()** method using the following attributes:

- tabletStartupScreen:     **/ftst/images/cf\_tablet\_startup.png**
- phoneStartupScreen:    **/ftst/images/cf\_phone\_startup.png**
- icon:                  **/ftst/images/cf\_phone\_icon.png**
- glossOnIcon:           **false**
- onReady :              **function() {}**

13. Inside the onReady function, declare a new variable named pnl that invokes a new Ext Panel with the following attributes:

- fullscreen: true
- html: 'Welcome to CrimeFinder'

14. Save the file. Your completed JavaScript should resemble the following:

```
<script type="text/javascript">

Ext.setup({
    tabletStartupScreen:
        '/ftst/images/cf_tablet_startup.png',
    phoneStartupScreen:
        '/ftst/images/cf_phone_startup.png',
    icon: '/ftst/images/cf_phone_icon.png',
    glossOnIcon: false,
    onReady: function() {
        var pnl = new Ext.Panel({
            fullscreen: true,
            html: 'Welcome to CrimeFinder'
        })
    }
})

</script>
```

15. On the Aptana toolbar, click on the **Run** button and select **Safari**
16. Once Safari launches you should see the HTML output of “Welcome to CrimeFinder”

– End of Walkthrough --

## Unit Summary



- Designing and developing applications for handheld devices is a completely different process than writing for desktop browsers.
- You will need to adapt your application design and information to accommodate handhelds with smaller screen resolutions and touch controls.
- Sencha Touch is an application framework built on web standards – HTML 5, JavaScript, and CSS3
- The Sencha Touch API enables you to control application layout, data access, input forms, multimedia output, and handle touch events.
- Sencha Touch ships as a ZIP archive that you can simply expand to an HTTP accessible directory on your web servers
- Your QA process involves testing with simulators as well as physical devices
- When you initialize an application, you should specify a startup image for phones and tablets as well as a launch icon
- Sencha Touch panels can be populated with HTML that is either encoded as JavaScript strings or available through an HTML DOM ID selector.

## Unit Review



1. What screen resolutions are typical for mobile devices?
2. Describe the process for testing an application hosted on your laptop on a physical device.
3. What types of user interface elements are supported by Sencha Touch?
4. What graphics resources are required for every Sencha Touch application?
5. What JavaScript and CSS resources are typically used by every Sencha Touch application?
6. How do you initialize a Sencha Touch application?

# Lab 2: Initializing the SubGenius University Application



During this lab you will develop the code necessary to start laying out the mobile web site for the fictional SubGenius University.

## Objectives

After completing this lab, you should be able to:

- Create an HTML 5 home page for the application
- Invoke the Ext.Setup() method

## Steps

1. Open Aptana Studio
2. Right-click on the **lab/lab2 folder** in the **ftst** project
3. Select **New > Other**
4. Click on **Web Files > HTML File**
5. Click **Next**
6. Enter a file name of **index.htm** and click **Finish**
7. Change the **<!DOCTYPE>** declaration to indicate that the page is using HTML5
8. Update the **<meta>** tag to the HTML5 standard
9. Change the document title to “**SubGenius University**”
10. Inside the **<head>** section of the document, insert a **<link>** tag that points to the Sencha Touch debug style sheet. Your code should appear similar to the following:

```
<link rel="stylesheet"
      href="/senchatouch/resources/css-debug/sencha-touch.css"
      type="text/css" />
```

11. After the code that you inserted from the previous step, insert a **<script>** tag to load the Sencha Touch debug API. Your code should appear similar to the following:

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

12. After the code that you inserted in the prior step, insert a **<script>** block.
13. Inside the **<script>** block, invoke the Ext.setup() method using the following attributes:
  - tabletStartupScreen: **/ftst/images/su\_tablet\_startup.png**
  - phoneStartupScreen: **/ftst/images/su\_phone\_startup.png**
  - icon: **/ftst/images/su\_icon.png**
  - glossOnIcon: **true**
  - onReady : **function() {}**
14. Inside the **onReady** function, declare a new variable named **pnl** that invokes a new Ext Panel with the following attributes:
  - fullscreen: **true**
  - contentEl: **welcomePage**
15. Save the file. Your completed JavaScript should resemble the following:

```
<script type="text/javascript">

Ext.setup({
    tabletStartupScreen:
        '/ftst/images/su_tablet_startup.png',
    phoneStartupScreen:
        '/ftst/images/su_phone_startup.png',
    icon: '/ftst/images/su_icon.png',
    glossOnIcon: false,
    onReady: function() {
        var pnl = new Ext.Panel({
            fullscreen: true,
            contentEl: 'welcomePage'
        })
    }
})

</script>
```

16. Inside the **<body>** section, insert a **<div>** tag with the following attributes:
  - id : welcomePage
17. Inside the **<div>** tag, add the following text:
  - SubGenius University Welcomes You!
18. Save the file
19. Test the file in Safari. You should see the HTML output of “SubGenius University Welcomes You!”
20. Browse the file in the Android simulator (Optional)

– End of Lab --

---

---

# **Unit 3:**

## **Defining Application Layout**

### **Unit Objectives**

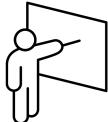
After completing this unit, you should be able to:

- Define your application flow and structure
- Create toolbars using buttons and icons
- Use animation to illustrate transitions

### **Unit Topics**

- Using Panels
- Defining Toolbars
- Implementing a Carousel Layout
- Implementing a Tabbed GUI
- Querying and Manipulating the DOM
- Using Sencha.io Src to Scale Images for Devices

## Using Panels



Panels act as basic containers for virtually any type of content that you wish to display. You can add button controls to either the top or the bottom of a panel and they may contain child components – including other panels. Most of the other Sencha Touch UI components inherit from Ext.Panel – including the Carousel, TabPanel, FormPanel, and NestedList classes.

As you discovered in the prior unit, the basic invocation syntax for a panel is the following:

```
var pnl = new Ext.Panel({
    fullscreen: true,
    html: 'Hello World'
})
```

### Populating a Panel with Content

A panel may be populated with content from four sources:

- HTML that is hardcoded within your Panel instantiation
- An HTML DOM block element
- Other Sencha Touch elements
- Data from a structured data set (**covered** in unit 5)

### Displaying HTML from your Panel instantiation

You can instantiate HTML into your panel as previously described by specifying a value for its HTML property as indicated by the following example:

```
var pnl = new Ext.Panel({
    fullscreen: true,
    html: 'Hello World'
})
```

### Re-purposing Content in an HTML DOM Element

You can reference content contained within your page's DOM by referencing its ID through the Panel.contentEl configuration option as illustrated by the following example:

```

...
<script type="text/javascript">

    Ext.setup({
        tabletStartupScreen:
            '/ftst/images/cf_tablet_startup.png',
        phoneStartupScreen:
            '/ftst/images/cf_phone_startup.png',
        icon: '/ftst/images/cf_icon.png',
        glossOnIcon: false,
        onReady: function() {
            var pnl = new Ext.Panel({
                fullscreen: true,
                contentEl: 'myContent'
            })
        }
    })

</script>
...
<body>
<div id="myContent">Hello World</div>
...
</body>

```

## Nesting Sencha Touch Containers

A Panel may contain any type of object based on the Ext.Component class. This includes, but is not limited to, toolbars, fieldsets, media, buttons, and other panels.

Specify embedded elements using the `items` attribute of the configuration object as indicated by the following example:

```

onReady: function() {

    var panel1 = new Ext.Panel({
        html: 'One'
    });

    var panel2 = new Ext.Panel({
        html: 'Two'
    });

    var myPanel = new Ext.Panel({
        fullscreen: true,
        items : [panel1,panel2]
    });
}

```

Output:

```

One
Two

```

## Defining Layouts

Typically when you nest UI elements you will need to define their layout properties. Otherwise, the default action is to simply render the items sequentially in the container, as demonstrated by the previous code example).

Layouts are specified as either strings or as objects through the layout attribute of the Panel configuration options as illustrated by the following code example:

```
onReady: function() {

    var panel1 = new Ext.Panel({
        html: 'One'
    });

    var panel2 = new Ext.Panel({
        html: 'Two'
    });

    var myPanel = new Ext.Panel({
        fullscreen: true,
        items : [panel1,panel2],
        layout: {
            type: 'hbox',
            align: 'left'
        }
    );
}
```

Valid layout type values may be one of the following:

- auto (default)
- card
- fit
- hbox
- vbox

You may specify additional attributes based on the layout type.

### Using the auto layout

The auto layout simply tiles your elements directly beneath each other.

It supports the a single additional configuration option:

Configuration Option	Description
itemCls	An optional extra CSS class that will be added to the container. This can be useful for adding customized styles to the container or any of its children using standard CSS rules

The following example illustrates using the “auto” layout type:

```

<style>
    .dashedBorder {
        border: 2px dashed red;
        padding: 5px;
        margin: 5px;
    }
</style>

...

onReady: function() {
    var panel1 =
        new Ext.Panel({ html: 'One' });
    var panel2 =
        new Ext.Panel({html: 'Two'});
    var myPanel = new Ext.Panel({
        fullscreen: true,
        items : [panel1,panel2],
        layout: {
            type: 'auto',
            itemCls: 'dashedBorder'
        }
    });
}

```

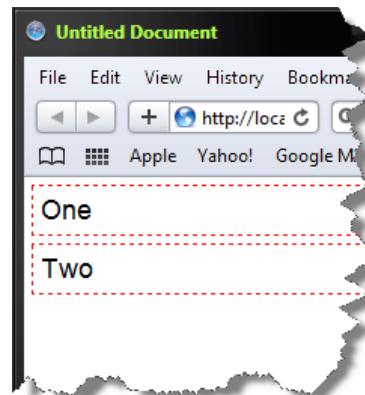


Illustration 1: Auto layout type

## Using the hbox Layout

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

Configuration options include the following:

Option	Description
align	<p>Specifies the vertical alignment of child components. Allowed values include:</p> <ul style="list-style-type: none"> <li>• <b>center</b> : align to the center of the container (default)</li> <li>• <b>end</b> : align to the bottom of the container</li> <li>• <b>start</b> : align to the top of the container</li> <li>• <b>stretch</b> : stretch components vertically to fill the container</li> </ul>

Option	Description
direction	Specifies the direction in which child components are laid out. Allowed values include: <ul style="list-style-type: none"> <li>• <b>normal</b> : Components are laid out in the order they are added (default)</li> <li>• <b>reverse</b> : Components are laid out in the opposite order in which they are added</li> </ul>
itemCls	An optional extra CSS class that will be added to the container. This can be useful for adding customized styles to the container or any of its children using standard CSS rules
pack	The horizontal alignment of child components. Allowed values include: <ul style="list-style-type: none"> <li>• <b>center</b> : Aligned to the center of the container</li> <li>• <b>end</b> : Aligned to the right of the container</li> <li>• <b>justify</b>: Justified with both the left and right of the container</li> <li>• <b>start</b> : Aligned to the left of the container</li> </ul>

The following code snippet and associated screen shot illustrate using the Hbox layout:

```
var myPanel = new Ext.Panel({
    fullscreen: true,
    items : [panel1,panel2],
    layout: {
        type: 'hbox',
        itemCls:
            'dashedBorder',
        align: 'start',
        direction: 'normal'
    }
});
```

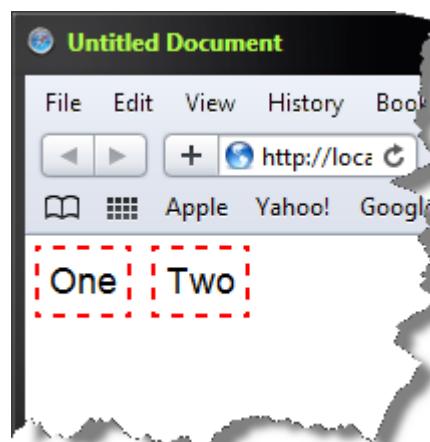


Illustration 2: Using hbox layout

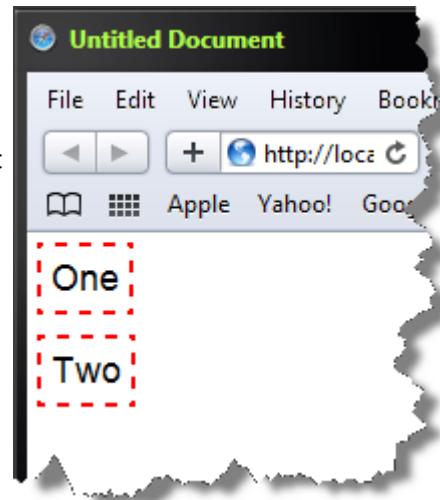
## Using the vbox Layout

The vbox layout is similar to the auto layout – it arranges elements vertically. Vbox layout supports the same configuration options as the previously documented hbox layout:

- align
- direction
- itemCls
- pack

The following example illustrates using a vbox layout:

```
var myPanel = new Ext.Panel({  
    fullscreen: true,  
    items : [panel1,panel2],  
    layout: {  
        type: 'vbox',  
        itemCls: 'dashedBorder',  
        align: 'start',  
        direction: 'normal'  
    }  
});
```



## Using the fit Layout

Fit is a base class for layouts containing a single item. It automatically expands the element to fill the layout's container as illustrated by the following example:

```
var myPanel = new Ext.Panel({  
    fullscreen: true,  
    items : [panel1],  
    layout: {  
        type: 'fit',  
        itemCls: 'dashedBorder'  
    }  
});
```

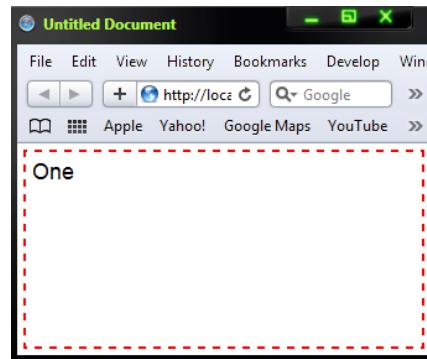


Illustration 4: Using the 'fit' layout type

## Using the card Layout

The card layout manages multiple child Components, each fit to the Container, where only a single child Component can be visible at any given time. This layout style is most commonly used for wizards, tab implementations, and the like.

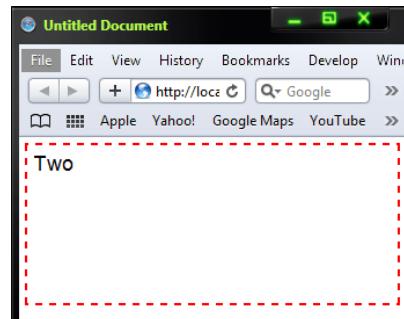
Card layouts support a single configuration option – itemCls.

Use the `setActiveItem()` method to set the index of the visible card. You can also specify an animation effect to be used during the transition of cards through the animation configuration option as indicated by the following example:

```
var myPanel = new Ext.Panel({
    fullscreen: true,
    items : [panel1,panel2],
    layout: {
        type: 'card',
        itemCls: 'dashedBorder',
        cardSwitchAnimation: 'fade'
    }
});
myPanel.setActiveItem(1,'wipe');
```

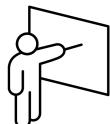
The following card animations are supported:

- fade
- slide
- flip
- cube
- pop
- wipe



**Illustration 5: The card layout**

## Defining Toolbars



The `dockedItems` configuration option for containers enables you to define toolbars or tab bars located at either the top, right, left, or bottom of a panel.

Toolbars can also be defined using the `Ext.Toolbar` class as illustrated by the following example:

```
var panel1 = new Ext.Panel({html: 'One'});
var panel2 = new Ext.Panel({html: 'Two'});

var handleClicks = function(b,e){
    if (b.getText() == 'Panel 1') {
        myPanel.setActiveItem(0);
    } else {
        myPanel.setActiveItem(1);
    }
}

var myToolbar = new Ext.Toolbar({
    dock: 'top',
    title: 'CrimeFinder',
    items: [
        {text: 'Panel 1', handler: handleClicks},
        {text: 'Panel 2', handler: handleClicks}
    ]
});

var myPanel = new Ext.Panel({
    fullscreen: true,
    items : [panel1,panel2],
    dockedItems: [myToolbar],
    layout: {
        type: 'card',
        itemCls: 'dashedBorder'
    }
});
```

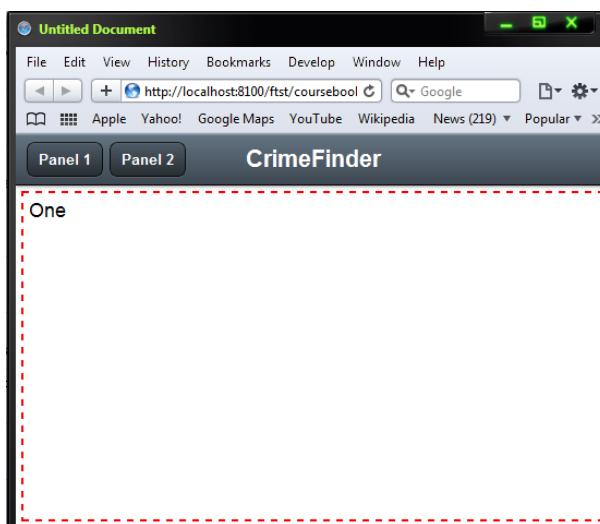


Illustration 6: A simple toolbar implementation

## Configuring the Toolbar

The Ext.Toolbar class extends the Ext.Container class, and therefore inherits its configuration options.

You will typically define the following toolbar configuration options:

Option	Description
dock	String. The location of the toolbar in a panel. Can be set to <b>top</b> (default), <b>bottom</b> , <b>left</b> , or <b>right</b>
title	String. The title of the toolbar
titleCls	String. The CSS class to apply to the title (defaults to 'x-toolbar-title')
items	Array. Typically an array of Ext.Button elements
disabled	Boolean. Defaults to false
ui	String. Style options for the Toolbar. Can be set to either <b>dark</b> (default) or <b>light</b>
layout	Object. A layout config object.

The following example docks two toolbars – at the top and bottom of a panel, by defining them in-line through the `dockedItems` attribute:

```
var myPanel = new Ext.Panel({
    fullscreen: true,
    items : [panel1,panel2],
    dockedItems: [
        {
            dock: 'top',
            xtype: 'toolbar',
            title: 'CrimeFinder',
            items: []
        },
        {
            dock: 'bottom',
            xtype: 'toolbar',
            title: '[buttons go here]',
            ui: 'light'
        }
    ],
    layout: { type: 'card', itemCls: 'dashedBorder' }
});
```



Illustration 7: Panel with docked toolbars

## Defining Toolbar Buttons

As previously stated, buttons are added to toolbars through the Ext.toolbar.items array configuration element. Button objects extend the Container class and their configuration options include, but are not limited to the following:

Config Option	Description
badgeText	String. The text to be used for a small badge on the button.
centered	String. Centers the button. Defaults to false.
handler	The function to be called when the button is pressed. The function is automatically passed the button object and the click event.
text	String. The html to be used as the button label
ui	String. Determines the look and feel of the button. Valid options are: normal (default), back, round, action, forward, and plain
icon	String. The path to an image to display in the button
iconAlign	String. Valid options are top, right, bottom, left (default)
iconCls	A css class which sets a background image to be used as the icon for the button
iconMask	Boolean. If set to true, it displays the icon associated with the iconCls value.
hidden	Boolean. Set it to true to initially hide the button. Defaults to false.

The following example illustrates the syntax for defining a button that uses a custom image along with different values for the iconAlign attribute:



**Illustration 8: Image alignment options**

```
items: [
    {text: 'VCard', icon: 'vcard.png', iconAlign: 'bottom'},
    {text: 'VCard', icon: 'vcard.png', iconAlign: 'left'},
    {text: 'VCard', icon: 'vcard.png', iconAlign: 'right'},
    {text: 'VCard', icon: 'vcard.png', iconAlign: 'top'}
]
```

## Configuring the Button UI

CSS button classes that ship with Sencha Touch include the following:

iconCls	Button	iconCls	Button	iconCls	Button
action		add		attachment	
bookmarks		bolt		chat	
compose		delete		home	
maps		organize		refresh	
reply		search		tag	

iconCls	Button	iconCls	Button	iconCls	Button
x-icon-mask trash		locate		favorites	
download		info		settings	
user					

Note: The complete set of icon images is located at:  
 [senchatouchdir]\resources\themes\images\default\pictos. You can use additional icons using the techniques covered in unit 7.

Use of the ui configuration options are illustrated below:



Illustration 9: Demonstration of ui configuration values

```
var toolbar_icons = {
  xtype: 'toolbar',
  dock: 'top',
  scroll: 'horizontal',
  items: [
    {iconMask: true, iconCls: 'home', ui: 'normal'},
    {iconMask: true, iconCls: 'home', ui: 'plain'},
    {iconMask: true, iconCls: 'home', ui: 'back'},
    {iconMask: true, iconCls: 'home', ui: 'round'},
    {iconMask: true, iconCls: 'home', ui: 'action'},
    {iconMask: true, iconCls: 'home', ui: 'forward'},
    {iconMask: true, iconCls: 'home', ui: 'normal',
      badgeText: '2'}
  ]
}
```

## Defining Segmented Buttons

Segmented buttons function in an analogous manner to radio buttons. By default, out of a series of defined buttons, only one can be in the “on” state at any given time. However, this behavior can be overridden so that multiple buttons can be in a pressed state simultaneously. The following example illustrates the syntax associated with defining these types of controls:

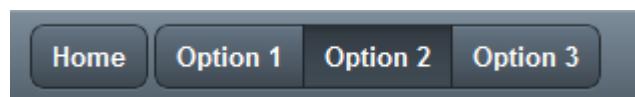


Illustration 10: A segmented button group

```
dockedItems: [
    {
        dock: 'bottom',
        xtype: 'toolbar',
        ui: 'light',
        items: [
            {text: 'Home'},
            {
                xtype: 'segmentedbutton',
                items: [
                    {text: 'Option 1', handler: tapHandler},
                    {text: 'Option 2', handler: tapHandler,
                     pressed: true},
                    {text: 'Option 3', handler: tapHandler},
                ]
            }
        ]
    }
]
```

## Centering Buttons

Use the layout property of the toolbar configuration options in order to center your buttons horizontally in a button bar as indicated by the following example:

```
dockedItems: [{{
    dock: 'bottom',
    xtype: 'toolbar',
    ui: 'light',
    layout: {pack :
              'center'},
    items: [
        {text: 'Home', iconCls: 'home'},
        {text: 'Info', iconCls: 'info'}
    ]
}}]
```

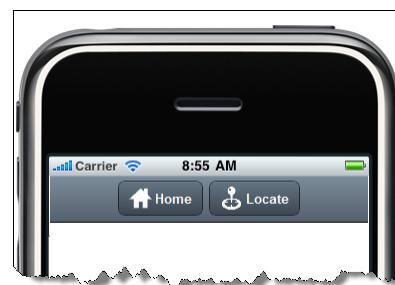


Illustration 11: Centered Buttons

## Walkthrough 3-1: Defining a Panel Layout



In this walkthrough, you will start the process of laying out the CrimeFinder mobile phone application by completing following tasks:

- Define a card layout using panels
- Define application toolbars
- Create event handlers for buttons

### Steps

#### Define the Panels

1. Open /walk/walk3-1/index.html in your editor
2. Where indicated by the comment, define the following panels:



Variable Name	HTML Contents
mapPnl	Insert Google Map Here
listPnl	Insert Crime List Here
feedsPnl	Insert RSS Feeds Here
videoPnl	Insert Video feeds here
reportPnl	Insert confess form here

3. Where indicated by the comment, define variable named **mainPnl** that occupies the entire screen and acts as a container for the panels that you defined in the previous step. Use a **card** layout and a **fade** animation.

4. Verify that your code appears similar to the following:

```
onReady: function() {  
  
    var mapPnl= new Ext.Panel ({html: 'Insert Google Map Here'});  
    var listPnl= new Ext.Panel({html: 'Insert Crime List here'});  
    var feedsPnl = new Ext.Panel({html: 'Insert RSS Feeds'});  
    var videoPnl = new Ext.Panel({html: 'Insert Cameras here'});  
    var reportPnl = new Ext.Panel({html: 'Insert form here'});  
  
    var mainPnl = new Ext.Panel({  
        fullscreen: true,  
        items: [mapPnl,listPnl,feedsPnl,videoPnl,reportPnl],  
        layout: {  
            type: 'card'  
        },  
        cardSwitchAnimation: 'fade'  
    })  
}
```

5. Save the file and browse it in Safari. You should see the text "Insert Google Map Here"

#### Add the Main Toolbar

6. Where indicated by the comment, declare a variable named mainTB that instantiates a new Toolbar that will be docked at the top of its container and centered horizontally. The toolbar should contain a single segmented button with the following attributes:

Button Text	Handler	Pressed
Map	tapHandler	TRUE
Events	tapHandler	FALSE
Feeds	tapHandler	FALSE
Video	tapHandler	FALSE
Confess	tapHandler	FALSE

7. Assign a value of 'maintoolbar' to the id property of mainTB

Your code should appear similar to the following:

```
var mainTB = new Ext.Toolbar({
    dock: 'top',
    id: 'maintoolbar',
    layout: {pack: 'center'},
    items: [
        {
            xtype: 'segmentedbutton',
            items: [
                {text: 'Map', handler: tapHandler, pressed: true },
                {text: 'Events', handler: tapHandler},
                {text: 'Feeds', handler: tapHandler},
                {text: 'Video', handler: tapHandler},
                {text: 'Confess', handler: tapHandler}
            ]
        }
    ]
});
```

8. Modify your **mainPnl** panel definition to include the toolbar that you defined from the prior step. Above the toolbar, insert a title of "CrimeFinder". Your mainPnl definition should appear as follows:

```
var mainPnl = new Ext.Panel({
    fullscreen: true,
    items: [mapPnl,listPnl,feedsPnl,videoPnl,reportPnl],
    layout: {
        type: 'card'
    },
    cardSwitchAnimation: 'fade',
    dockedItems: [
        {
            xtype: 'toolbar',
            dock: 'top',
            title: 'CrimeFinder'
        },
        mainTB
    ]
})
```

#### Define the button click handler

9. Where indicated by the comment, define a function named **tapHandler** that accepts two arguments named b and e.
10. Inside the function, insert a switch block that evaluates the text of the button that was clicked. Your code should appear similar to the following:

```
var tapHandler = function(b,e) {
    switch (b.getText()){

    }
}
```

11. Inside the switch() block that you inserted in the prior step, evaluate the button labels on the toolbar and set focus to the appropriate panel. Your code should appear as follows:

```
case 'Map': mainPnl.setActiveItem(0); break;
case 'Events': mainPnl.setActiveItem(1); break;
case 'Feeds': mainPnl.setActiveItem(2); break;
case 'Video': mainPnl.setActiveItem(3); break;
case 'Confess': mainPnl.setActiveItem(4); break;
```

12. Save the file and test it. You should be able to click on the buttons to toggle the visibility of your panels.

### Enhance the Crime Listing Panel

13. Expand your definition for **IstPnl** to include a toolbar that is docked at the bottom of the panel. The toolbar should have a single segmented button consisting of two icon and text-based buttons as indicated by the following table:

text	iconCls	iconMask
Type	organize	TRUE
Date	organize	TRUE

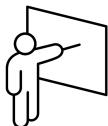
The entire **IstPanel** definition should resemble the following following:

```
var listPnl = new Ext.Panel({
    html: 'Insert Crime List here',
    dockedItems: [
        { xtype: 'toolbar',
            dock: 'bottom',
            layout: {pack: 'center'},
            items: [
                {xtype: 'segmentedbutton',
                    items: [
                        {text: 'Type', iconCls: 'organize', iconMask: true},
                        {text: 'Date', iconCls: 'organize', iconMask: true}
                    ]
                }
            ]
        }
    ]
});
```

14. Save the file and test.

– End of Walkthrough –

# Implementing a Carousel Layout



Carousels are customized Panels that enable you to slide back and forth between different child items. You can define either horizontal or vertical scrolling carousels. You can define them using the Ext.Carousel constructor as indicated by the following code example:

```
var carousel = new Ext.Carousel({
    direction: 'horizontal',
    indicator: true,
    ui: 'dark',
    items: [
        {
            xtype: 'panel',
            html: '<h1>Carousel</h1>',
            cls: 'card1'
        },
        {
            xtype: 'panel',
            html: '2',
            cls: 'card2'
        },
        {
            xtype: 'panel',
            html: '3',
            cls: 'card3'
        }
    ]
});
```



**Illustration 12: Horizontal Carousel**

*In the previous example, the xtype declaration is optional since it is the item default.*

Carousel-specific configuration properties include the following:

Property	Description
baseCls	String. The base CSS class to apply to the Carousel's element (defaults to 'x-carousel').
direction	String. The direction of the Carousel. Default is 'horizontal'. 'vertical' also available.
indicator	Boolean. Provides an indicator while toggling between child items to let the user know where they are in the card stack.
ui	String. Style options for Carousel. Default is 'dark'. 'light' is also available.

## Walkthrough 3-2: Defining a Carousel Layout



In this walkthrough, you will continue the process of laying out the CrimeFinder mobile phone application, depicted at right, by defining a carousel layout for the Video section.



### Steps

#### Define the Carousel

1. Open /walk/walk3-2/index.html in your editor
2. Comment out the videoPnl declaration that you added from the prior walkthrough
3. Where indicated by the comment, redeclare the videoPnl variable as a carousel. Your syntax should appear as follows:
 

```
var videoPnl = new Ext.Carousel( );
```
4. Inside the Ext.Carousel constructor, add a configuration object that sets the direction to horizontal and defines two panels with the following properties:

html	cls
Video 1 goes here	card1
Video 2 goes here	card 2

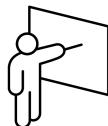
Your complete videoPnl declaration should appear similar to the following:

```
var videoPnl = new Ext.Carousel({
direction: 'horizontal',
items: [
  {xtype: 'panel', html: 'Video 1 goes here', cls: 'card1'},
  {xtype: 'panel', html: 'Video 2 goes here', cls: 'card2'}
]
});
```

5. Save the file and test it using multiple simulators. You should see the carousel appear under the "Video" heading.

– End of Walkthrough –

## Implementing a Tabbed GUI



The TabPanel is a customized Panel that enable you to press buttons to slide between different content views. You can define the animation to be used during the card transition, enable sorting for the tab bar, and define the position of the tab bar. TabPanels can defined using the Ext.TabPanel constructor as indicated by the following code example:

```
new Ext.TabPanel({
    fullscreen: true,
    ui: 'dark',
    sortable: true,
    items: [{
        title: 'Tab 1',
        html: '1',
        cls: 'card1'
    }, {
        title: 'Tab 2',
        html: '2',
        cls: 'card2'
    }, {
        title: 'Tab 3',
        html: '3',
        cls: 'card3'
    }]
});
```

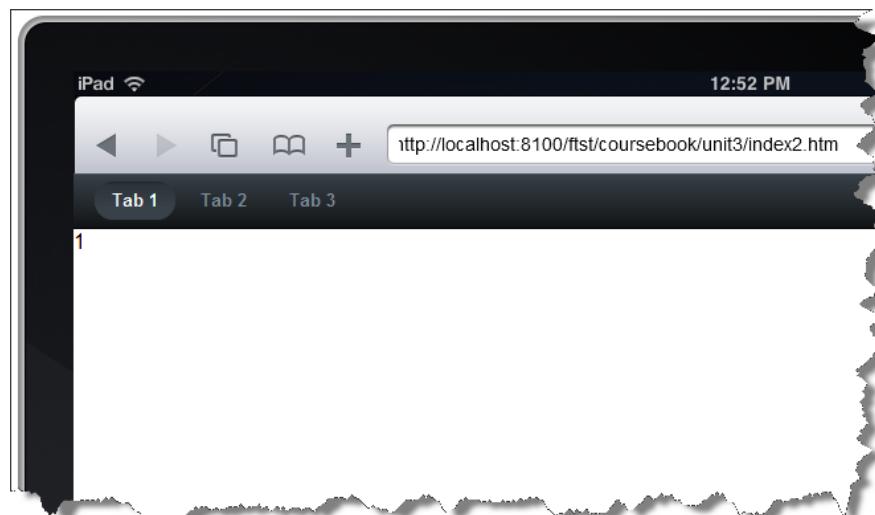


Illustration 13: Tabs running in the iPad simulator

## Using TabPanel Configuration Options

Tabpanel-specific configuration options include the following:

Configuration Option	Description
animation	String. Animation to be used during card transition. Valid options include <code>slide</code> (default), <code>fade</code> , <code>flip</code> , <code>cube</code> , <code>pop</code> , and <code>wipe</code>
sortable	Boolean. Enables sorting functionality.
tabBar	Object. An Ext.TabBar configuration
tabBarPosition	String. Where to dock the tab bar. Valid options are <code>top</code> and <code>bottom</code> .
ui	String. Valid options are <code>light</code> and <code>dark</code> (default)

## Styling the Tab Bar

You can configure the tab bar through a combination of the `Ext.TabBar` configuration object and the individual panel definitions of the `TabPanel`. Tab buttons are implemented through the `Ext.tab` class, which in turns inherits from the `Ext.Button` class. Therefore, the following configuration options are available to style each tab:

Config Option	Description
badgeText	String. The text to be used for a small badge on the button
icon	String. The path to an image to display in the button
iconCls	String. A css class which sets a background image to be used as the icon for this button
text	String. The text to display on the tab.
ui	String. The look and feel of the button. Valid options include <code>normal</code> (default), <code>back</code> , <code>round</code> , <code>action</code> , and <code>forward</code> .

The following example illustrates defining a tabbed layout with stylized buttons:

```
onReady: function() {
    mytabs = new
    Ext.TabPanel({
        fullscreen: true,
        ui: 'dark',
        sortable: true,
        tabBar: {
            dock: 'bottom',
            layout: {
                pack: 'center'
            }
        },
        animation: {
            type: 'cardslide',
            cover: true
        },
        items: [
            { title: 'Tab 1', html: '1', iconCls: 'info' },
            { title: 'Tab 2', html: '2', iconCls: 'favorites' },
            { title: 'Tab 3', html: '3', iconCls: 'user' }
        ]
    });
}
```

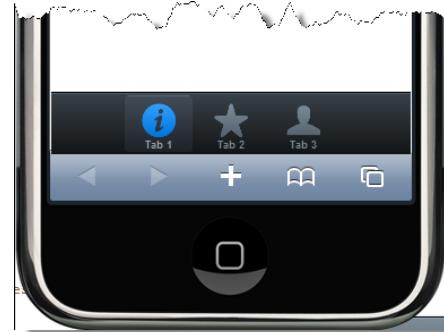


Illustration 14: Bottom-centered tab bar with icons

## Walkthrough 3-3: Defining a Tab Bar



In this walkthrough, you will continue the process of laying out the CrimeFinder mobile phone application, depicted at right, by defining a tab bar for the RSS feeds section.



### Steps

#### Define the Tab Panel

1. Open /walk/walk3-3/index.html in your editor
2. Modify the constructor for the feedsPnl variable to invoke the Ext.TabPanel constructor.
3. Delete the contents of the configuration object from the Ext.TabPanel constructor and set the following attributes:
  - ui: 'dark'
  - sortable: 'true'
  - fullscreen: true
4. Add a tabBar attribute to the configuration object with the following attributes:
  - dock: 'bottom'
  - layout: {pack: 'center'}
5. Add the following items to the tab panel configuration object:

**Illustration 15: The result of the walkthrough**

<b>title</b>	<b>iconCls</b>	<b>html</b>
Missing Kids	team	Insert FBI RSS Missing Children here
Top 10	team	Insert 10 most wanted here

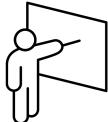
6. Save the file and test in your simulators.

Your code should appear similar to the following:

```
var feedsPnl = new Ext.TabPanel({
    fullscreen: true,
    ui: 'dark',
    sortable: true,
    tabBar: {
        dock: 'bottom',
        layout: {pack: 'center'}
    },
    items: [
        {
            title: 'Missing Kids',
            iconCls: 'team',
            html: 'Insert FBI RSS Feed here'
        },
        {
            title: 'Top 10',
            iconCls: 'team',
            html: 'Insert 10 most wanted'
        }
    ]
});
```

-- End of Walkthrough –

## Querying and Manipulating the DOM



You can query for Component objects using two different mechanisms:

- Ext.ComponentManager enables you to use syntax similar to CSS selectors
- Component methods up(), down(), prev(), and next() are convenient to use in the event handlers of children and parent component items.

Components generate HTML markup, which you can also access and manipulate via a set of element methods.

### Using Ext.ComponentQuery

Sencha Touch enables you to search for components either globally or within a specific Ext.container.Container on the document using a similar syntax to a CSS selector.

Components can be retrieved by using their xtype with an optional . prefix, i.e.:

- component or .component
- panel or .panel

An itemId or id must be prefixed with a #

- #myContainer

Attributes must be wrapped in brackets

- component[autoScroll]
- panel[title="Test"]

Ext.ComponentQuery returns an array of components. For example:

```
// retrieve all Ext.Panels that are children of the
// container with an id of viewport
var aPanels = Ext.getCmp('viewport').query('panel');

// retrieve all tabpanels and toolbars
var aToolbarLists =
    Ext.getCmp('viewport').query('tabpanel, toolbar');

// retrieve all select fields with a name of "myfield"
var cmpSelect =
    Ext.getCmp('viewport').query('selectfield[name="myfield"]');
```

## Using Tests in your Component Query

Member expressions from candidate Components may be tested. If the expression returns a *truthy* value, the candidate Component will be included in the query:

```
var disabledFields = myFormPanel.query("{isDisabled()}");
```

## Using DOM Pseudo Classes

Pseudo classes may be used to filter results in the same way as in DomQuery:

```
// Function receives array and returns a filtered array.
Ext.ComponentQuery.pseudos.isZero = function(items) {
    var i = 0, l = items.length, c, result = [];
    for (; i < l; i++) {
        if (!(c = items[i]).getValue() === 0) {
            result.push(c);
        }
    }
    return result;
};

var zeroFields = myFormPanel.query('field:isZero');

if (zeroFields.length) {
    // some action
}
```

## Using the up() and down() methods

The `up()` method walks up the `ownerCt` axis looking for an ancestor Container which matches the passed simple selector.

`up()` accepts a single parameter – a string value that indicates the selector to test for. It returns back the first matching ancestor container, or `undefined` if no match was found.

For example if a form field was defined as a child within atabpanel, the following code would return the form field's owner:

```
var owningTabPanel = formfield.up('tabpanel');
```

The `down()` method searches through a container's items and returns the first matching component or `undefined` if no match is found.

The following example illustrates using these two methods:

```
var pnl    = button.up('panel'); // get button container
var form   = pnl.down('form'); // get form child of panel
```

## Using the previousSibling() and nextSibling() methods

The `previousSibling()` and `nextSibling()` methods return components that are adjacent to the referenced object. Both methods accept an optional component query selector. The functions return `null` if no sibling items match. You can abbreviate their invocation using the `prev()` and `next()` methods.

The following example illustrates using the methods:

```
var aPanels = Ext.ComponentQuery.query('panel');
var sbtn = aPanels[0].down('splitbutton');
var nextBtn = sbtn.nextSibling();
var prevBtn = sbtn.previousSibling();
```

*Note: These methods can also be invoked as `prev()` and `next()`*

## Adding and Removing Components

You can add and remove components by using the following methods:

- **add** (object/Array component) : Ext.Component  
Adds a new component to a container.  
  
In addition, invoking this method also performs the following actions:
  - It fires the `beforeadd` event before adding
  - The Container's default config values will be applied accordingly
  - It fires the `add` event after the component has been added.
- **insert** (Number index, Ext.Component component) : Ext.Component  
Inserts a component into the container at the specified numeric index.  
Invoking this method triggers the same events as previously described by the `add()` method.
- **remove** (Component/String component, bool Autodestroy) : Ext.Component  
Removes the specified component from the container

In addition, invoking this method also performs the following actions:

- It fires the `beforeremove` event before removing
- It fires the `remove` event after the component has been removed

The following snippet illustrates using Component methods to add a new button to a segmentedbutton component:

```
mainTB.down('segmentedbutton').insert(0,{  
    xtype: 'button', text: 'hey now'});
```

*Note: You will need to invoke the container's `doLayout()` method if the component had been rendered prior to invoking the `add()` or `insert()` methods.*

## Walkthrough 3-4: Traversing Components



In this walkthrough, you will perform the following tasks:

- Use Ext.ComponentQuery
- Use the up(), down(), prev() and next() methods to walk through the component hierarchy
- Use a DOM element method to change the HTML of a button.

### Steps

#### Experiment with Component Traversal

1. Open **/walksolution/walk6-2/index.html** in your browser.
2. Open your browser's debugger
3. In the Javascript console, define a variable named **mainTB** that points to the persistent toolbar in the application.

```
var mainTB = Ext.getCmp('mantoolbar')
```

4. Use the down() method to retrieve a pointer to the first button that was defined in the toolbar and inspect its properties with your instructor.

```
mainTB.down('button');
```

5. Use the up() method to retrieve the parent segmented button of the component that you retrieved in the previous step.

```
mainTB.down('button').up('segmentedbutton');
```

#### Experiment with Component Manipulation

6. Use the remove() method to delete the first button

```
mainTB.down('segmentedbutton').remove(0);
```

7. Use the segmentedbutton.insert() method to insert a new button at the beginning of the splitbutton component.

```
mainTB.down('segmentedbutton').insert(0, {  
    xtype: 'button', text: 'hey now'});
```

8. Invoke the doLayout() method of the segmentedbutton in order to render the component that you inserted from the prior step.

```
mainTB.down('segmentedbutton').doLayout()
```

9. Use the next() method to retrieve a reference to the button adjacent to the component that you inserted from the prior step.

```
mainTB.down('segmentedbutton').down('button').next()
```

### Work with Elements

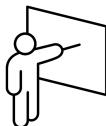
10. Use the getEl() method to retrieve the HTML that was generated by the segmentedbutton.

```
mainTB.down('segmentedbutton').getEl();
```

11. Review the Ext.Element methods with your instructor.

– End of Walkthrough --

## Using Sencha.io Src to Scale Images for Devices

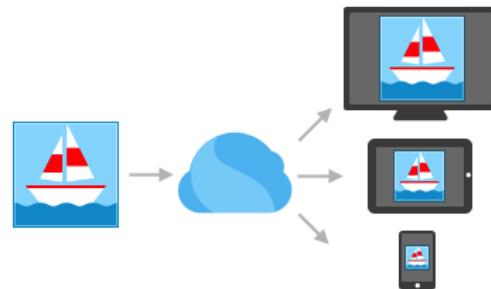


Sencha.io Src sizes your images to the device that is requesting them, then caches and optimizes them for efficient repeat delivery. With Sencha.io src, you don't need to worry about scaling your image assets for the variety of mobile screens. Simply place a single high resolution image on your server, point your `<img>` tag to Sencha.io Src and let the cloud take care of the rest.

Sencha.io Src is essentially a proxy that lies between image assets (hosted either on your own server or by a third party) and the browser or application requesting them via HTTP. The API is accessed entirely via placing a prefix before the original image URL. This prefix gives you declarative access to all of the different types of transformation that the service can perform. This approach makes the service very easy to add to existing web sites or apps without any programming knowledge.

Sencha.io Src supports the following features:

- Defined Sizing
- Formulaic Sizing
- Percentage Sizing
- File Format Conversion
- Domain Sharding
- Data Proxy



**Illustration 16: Scale your images dynamically with Sencha.io Src**

## Auto-Resizing an Image for a Mobile Device

Sencha.io Src will resize the image to fit the physical screen of the mobile handset visiting your site, based on its user-agent string. For example, if an iPhone 3GS visits the site, the image will be constrained to its screen size of 320px × 480px. Aspect ratios are always preserved.

Simply replace the URL of your publicly hosted image:

```
<img  
    src='http://sencha.com/files/u.jpg'  
    alt='My large image'  
/>
```

with the following:

```
<img  
    src='http://src.sencha.io/http://sencha.com/files/u.jpg'  
    alt='My smaller image'  
/>
```

## Using Defined Sizing

As indicated by the following example, you can resize an image to a specific width and height by passing additional width/height arguments to Sencha IO.

```
<img  
    src='http://src.sencha.io/320/200/http://sencha.com/files/u.jpg'  
    alt='My constrained image'  
    width='320'  
    height='200'  
/>
```

## Resizing Images by Percentage

Similarly, if you want to scale the graphic to a proportion of the screen, use the x prefix. The value provided is interpreted as an integer percentage from 1 to 100.

```
<img  
    src='http://src.sencha.io/x50/http://sencha.com/files/u.jpg'  
    alt='My image, constrained by half the width of the screen'  
/>
```

## Using Sencha IO as a JSON-P Data Proxy

With Sencha IO you can request that images in base-64 encoded form and have that content passed back to the JavaScript method of your design. From there you could programatically manipulate the data, cache it into local storage, or set it as the SRC of an <IMG> tag in your application. The callback function is provided as a dotted-suffix to the 'data' segment.

For example:

```
http://src.sencha.io/data.myCallBack/http://sencha.com/files/u.jpg
```

Returns the following JavaScript response:

```
myCallBack('data:image/jpeg;base64,/9j/4AAQSkZJRgABAQ...');
```

You may encounter scenarios where you want to run the callback multiple times with some sort of identifier, attaching it to different <img> elements throughout the document.. Hyphens in your callback name are used as separators, and subsequent portions get treated as successive string arguments. (Also, the callback name itself can contain dotted syntax so you can invoke a method within an object or a function within a specific namespace).

For example, the following URL:

```
http://src.sencha.io/data.MyApp.myCallBack-img2/http://sencha.com/files/u.jpg
```

Returns the following result::

```
MyApp.myCallBack('img2','data:image/jpeg;base64,/9j/4AAQSkZJQ...');
```

## Walkthrough 3-5: Using Sencha.io Src



In this walkthrough, you will perform the following tasks:

- Use Sencha.io Src to change the scale of an image
- Use Sencha.io Src to return an image in base-64 encoded format

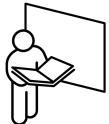
### Steps

#### Experiment with Sencha IO

1. Open **Safari**
2. Type the following url:  
<http://www.senchatraining.com/ftst/images/capitol.jpg>
3. Note the size of the file
4. Change the URL to the following:  
<http://src.sencha.io/http://senchatraining.com/ftst/images/capitol.jpg>
5. Note the size of the file
6. In Safari, select Develop > User Agent > Safari iOS 4.1 iPhone
7. Note that the file has automatically shrunk
8. Change the URL to the following:  
<http://src.sencha.io/320/200/http://senchatraining.com/ftst/images/capitol.jpg>
9. Note that the image has been resized
10. Change the URL to the following in order to return the image data base-64 encoded via a Javascript callback:  
<http://src.sencha.io/data.myCallback/http://senchatraining.com/ftst/images/capitol.jpg>

– End of Walkthrough --

## Unit Summary



- Ext.Panel serves as your foundation for content layout and display
- With Ext.Panel, HTML content can be displayed through either the html or contentEl configuration options
- Apply the x-hidden CSS class to any HTML DOM element referenced by the contentEl attribute.
- Panels have five different layout configurations: auto, fit, card, hbox, and vbox
- Center elements in a panel using the layout.pack option.
- Toolbars extend Ext.Panel and can be instantiated either directly through the Ext.Toolbar constructor, or implicitly through the dockedItems attribute of the Panel configuration object
- There are a large number of button styles and icons that ship with Sencha Touch
- You can use your own .png files to instantiate custom buttons
- By default, segmented buttons exhibit a mutual exclusivity behavior
- Carousels are customized Panels that enable you to slide back and forth between different child items.
- TabPanel buttons may be highly customized
- Virtually any content that can be rendered inside of a <div> may be displayed in a Panel

## Unit Review



1. What is the name of the method that enables you to programmatically set the active card in a layout?
2. Which two configuration attributes enable you to display a toolbar button using a built-in Sencha touch button style?
3. Which Ext classes can you use to lay out your application?
4. The carousel layout can only be used to slide panels horizontally (true/false)
5. You cannot have two carousels visible on the screen at the same time (true/false)
6. What six types of card transition animation are supported by directly Sencha touch?

## Lab 3: Layout the SubGenius University Application



During this lab you will implement the layout of the mobile web site for the fictional SubGenius University. The site has the following sections:

- **Welcome**  
Contains a three-position carousel. The first page contains a welcome message, the second page contains a list of instructors, and the third page will contain a Google map (Unit 6)
- **Courses**  
Display a nested list of courses read from a database (unit 5) and a video introducing specific classes (unit 6)
- **Schedule**  
Displays a simple list of training classes (unit 5)
- **Enroll**  
An enrollment form (unit 4)

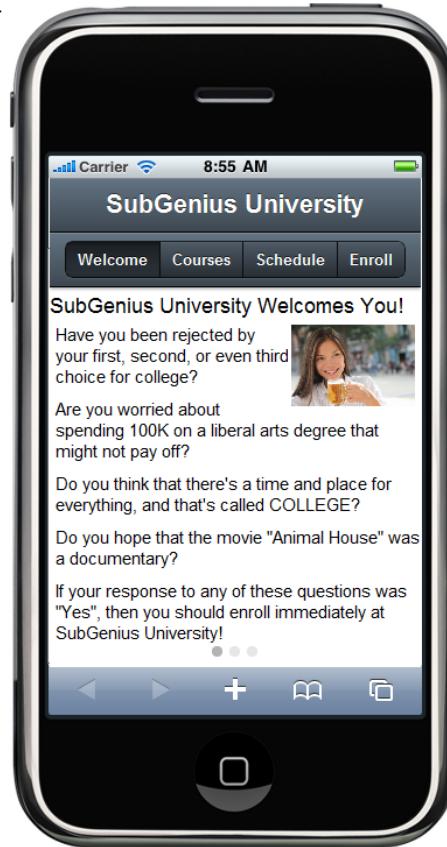


Figure 1: The SubGenius University Application

## Objectives

After completing this lab, you should be able to:

- Define panels
- Deploy a toolbar
- Implement a carousel layout

## Steps

1. Open your editor
2. Open /lab/lab3/index.htm and review its contents. Note that this is similar to the solution that you created in lab 2.

### Define the Main Panel

3. Where indicated by the comment, define a new panel named mainPnl with the following attributes:
  - fullscreen: true
  - layout : { type : 'card'}
  - cardSwitchAnimation: 'pop'
4. Define two toolbars that are both docked at the top of the panel. Review the screen shot on the previous page as a guide for how the toolbars should appear. The top toolbar should simply consist of the title "SubGenius University". The second toolbar should have a single, centered, segmented button with the following options:
  - Welcome
  - Courses
  - Schedule
  - Enroll
5. Save the file and browse. Your code should appear similar to the following:

```
var mainPnl = new Ext.Panel({
    fullscreen: true,
    layout: {type: 'card'},
    cardSwitchAnimation: 'pop',
    dockedItems: [
        {
            xtype: 'toolbar',
            dock: 'top',
            title: 'SubGenius University'
        },
        {
            xtype: 'toolbar',
            dock: 'top',
            layout: {pack: 'center'},
            items: [{
                xtype: 'segmentedbutton',
                items: [
                    {text: 'Welcome, pressed: true'},
                    {text: 'Courses'},
                    {text: 'Schedule'},
                    {text: 'Enroll'}
                ]
            }]
        }
    ]
})
```

### Define the Welcome Panel

6. Where indicated by the comment, define a Carousel panel named welcomePnl with the following attribute:
  - direction: 'horizontal'

7. Add panels to your Carousel that output the contents of the three <div> tags in the <body> section of your page.
8. Check that your Carousel panel definition resembles the following:

```
var welcomePnl = new Ext.Carousel({  
    direction: 'horizontal',  
    items: [  
        {xtype: 'panel', contentEl: 'welcomePage'},  
        {xtype: 'panel', contentEl: 'instructorsPage'},  
        {xtype: 'panel', contentEl: 'locations'}  
    ]  
});
```

9. Add the **welcomePnl** to the **items** array of your **mainPnl** definition
10. Save the file and browse. You should see the carousel control and drag-slide horizontally to be able to view each of its panels.

### Define the Courses Panel

11. Where indicated by the comment, define a new Panel named **coursesPnl** with the following attribute:
  - html: 'Insert Course List here'
12. Add the **coursesPnl** to the **items** array of your **mainPnl** defintion.

### Define the Segmented Button Handler

13. Modify your segmentedbutton, located in the **mainPnl** definition, to invoke a JavaScript function named **tapHandler** when the user clicks on any of the buttons.
14. Where indicated by the comment, define a function named **tapHandler** that accepts two arguments named **b** and **e**.
15. Inside the **tapHandler** function, insert a **switch()** block that evaluates the text of the button that was pressed.
16. Based on the button that was pressed, set the appropriate screen to display. Your code should appear similar to the following:

```
var tapHandler = function(b,e) {  
    switch (b.getText()) {  
        case 'Welcome':  
            mainPnl.setActiveItem(0);  
            break;  
        case 'Courses':  
            mainPnl.setActiveItem(1);  
            break;  
        case 'Schedule':  
            mainPnl.setActiveItem(2);  
            break;  
        case 'Enroll':  
            mainPnl.setActiveItem(3);  
            break;  
    }  
}
```

17. Save the file and test. Clicking on the **Courses** button should bring forward the **Courses** panel.

### Create the Schedule Panel

18. Where indicated by the comment, define a new panel named **schedulePnl** that contains the following attribute:

```
html:  
    'Insert list of classes here'
```

19. Add a toolbar, docked at the bottom of the panel with two buttons labeled “Sort By Date” and “Sort by Name”.
20. Add the **schedulePnl** to the **items** array of your **mainPnl** definition.
21. Save the file and browse. Click on the Schedule button. Your screen should look like the screenshot at right.
22. Verify that your code resembles the following:



```
var schedulePnl = new Ext.Panel({  
    html: 'Insert list of classes here',  
    dockedItems: {  
        xtype: 'toolbar',  
        layout: {pack: 'center'},  
        dock: 'bottom',  
        items: [  
            {  
                xtype: 'segmentedbutton',  
                items: [  
                    {text: 'By Date',  
                     pressed: true, iconCls: 'organize', iconMask: true},  
                    {text: 'By Name',  
                     iconCls: 'organize', iconMask: true}  
                ]  
            }  
        ]  
    }  
})
```

### Define the Enrollment Form Panel

23. Where indicated by the comment, define a panel named **enrollPnl** that has the following attribute:
- html: 'Insert form here'
24. Add the **enrollPnl** to the **items** array of your **mainPnl** definition
25. Save the file and test.

– End of Lab --

(This page intentionally left blank)

---

---

# **Unit 4:**

## **Working with Forms**

### **Unit Objectives**

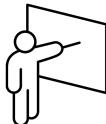
After completing this unit, you should be able to:

- Create a form
- Use form elements to capture input
- Add validation to form fields
- Dynamically change a form layout based on user interaction
- Pre-fill a form with data
- Submit form data to a server

### **Unit Topics**

- Defining an Input Form
- Handling Form Events
- Loading Data from a Model
- Submitting Form Data to a Server

## Defining an Input Form



You can define data entry forms using the Ext.form.FormPanel constructor. Sencha Touch supports most HTML 5 input field types including the following:

- Check Box
- Date Picker
- Email Field
- Fieldset
- Hidden
- Number
- Password
- Radio Buttons
- Search
- Select
- Slider
- Spinner
- Text
- Text Area
- Toggle
- URL

Instantiating a new form panel is similar to defining a panel that contains child elements:

```
var form = new Ext.form.FormPanel({  
    items: [  
        {xtype: 'textfield', name: 'fname', label: 'First'},  
        {xtype: 'textfield', name: 'last', label: 'Last'}  
    ]  
});
```

## Reviewing the Field class

Most form elements extend the Ext.form.Field class and therefore inherit the following configuration properties:

Property	Description
autoCapitalize	Boolean. If set to true, the first letter input is capitalized.
autoComplete	Boolean. If set to true (default), enables automatic completion of input
autoCorrect	Boolean. If set to true (default), automatically corrects misspellings.

Property	Description
autoCreateField	Boolean. Set to True (default) to automatically create the field input element on render. Set to false for any Ext.Field subclasses that don't need an HTML input (e.g. Ext.Slider and similar)
autoFocus	Boolean. Set to true to place the cursor into the form field on render. Defaults to false.
cls	String. A custom CSS class to apply to the field's underlying element. Defaults to "".
disabled	Boolean. Set to true to disable the field. Defaults to false. Note that disabled fields are not submitted in an HTTP Post.
hasFocus	Boolean. Set to true to set input focus immediately after rendered. Defaults to false.
id	Set the ID of the component
inputType	The type attribute for input fields – e.g. <code>radio</code> , <code>text</code> (default), <code>password</code> , <code>file</code> , <code>checkbox</code> . You must use this attribute to support file uploads.
label	String. The label to associate with the field
labelAlign	String. The location to render the label. Valid options are <code>top</code> and <code>left</code> (default)
labelWidth	String. The width of the label. Defaults to 30%
maxLength	Integer. The maximum allowed number of input characters.
name	The input field's <code>name</code> attribute.
placeHolder	String. The value to display in the field when the input is empty.
required	Boolean. Set to true to make the input required. Note that this does not prevent form submission if the field is left empty.
tabIndex	Integer. The tab index for the field. This only applies to fields that are rendered.
value	The default value for the field.

## Defining Text Fields

Text fields are typically declared as part of the Ext.form.FormPanel constructor as demonstrated by the following code example:

```
var formPnl = new Ext.form.FormPanel({
    items: [
        {
            xtype: 'textfield',
            name: 'lname',
            label: 'Last Name',
            required: true,
            maxlength: 20,
            autoFocus: true
        },
        {
            xtype: 'textfield',
            name: 'fname',
            label: 'First Name',
            required: true,
            maxlength: 20
        }
    ]
});
```



Figure 1: Text input fields in iOS

Note that you can also instantiate a text field using the Ext.form.Text constructor.

Ext.form.Text does not have any class-specific configuration options. It extends the Field and Component classes.

## Defining Telephone Number Input Fields

Telephone fields are not directly supported by Sencha Touch, however, you can easily deploy them by setting the inputType attribute of textfield to tel as indicated by the following example. Note that the tel field will activate a special onscreen keyboard.

```
{
    xtype: 'textfield',
    name: 'phone',
    label: 'Phone',
    required: true,
    inputType: 'tel'
}
```



Figure 2: Using the inputType property to define a telephone field

## Defining Password Fields

The `PasswordField` extends the `TextField` class. At run time, it functions almost identically to a text field except that data entry is masked.

You can instantiate password fields explicitly either through the `Ext.form.PasswordField` constructor or implicitly through an `Ext.form.FormPanel` configuration object as indicated below:

```
var formPnl = new Ext.form.FormPanel({
    items: [
        {
            xtype: 'textfield',
            name: 'username',
            label: 'User Name',
            required: true,
            maxlength: 20,
            autoFocus: true
        },
        {
            xtype: 'passwordfield',
            name: 'password',
            label: 'Password',
            required: true,
            maxlength: 20
        }
    ]
});
```



Figure 3: Password masking in iOS

## Implementing a Search Field

The SearchField (Ext.form.Search) extends Ext.form.Field. It wraps an HTML5 search input control. Typically you would use this as an interface to provide custom search services to your application. It does not have any class-specific configuration options.

The following example illustrates adding a search box to an application toolbar:

```
<style>
.leftToolbarTitle {
    position: absolute;
    top: 0;
    padding-left: 5px;
    bottom: 0;
    left: 0;
    z-index: 0;
    line-height: 2.1em;
    font-size: 1.2em;
    text-align: center;
    font-weight: bold;
    color: white;
}
</style>
...
var myPanel = new Ext.Panel({
    fullscreen: true,
    items : [panel1,panel2],
    layout: {
        type: 'card',
        extraCls: 'dashedBorder'
    },
    dockedItems: [{{
        dock: 'top',
        xtype: 'toolbar',
        title: 'CrimeFinder',
        titleCls: 'leftToolbarTitle',
        items: [
            {xtype: 'spacer'},
            {
                xtype: 'searchfield',
                name: 'searchquery',
                width: 100,
                placeHolder: 'Search'
            }
        ]
    }}];
});
```



Figure 4: Search field in iOS

## Defining Email Fields

The Email field extends the TextField class. At run time, it functions almost identically to a text field with the following two key distinctions:

- Data is validated to ensure proper formatting
- The device's onscreen keyboard is changed to facilitate data entry

You can instantiate email fields explicitly either through the Ext.form.Email constructor or implicitly through an Ext.form.FormPanel configuration object as indicated below:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield',
                name: 'name',
                label: 'Full Name',
                required: true ,
                autoFocus: true
            },
            {
                xtype: 'emailfield',
                name:'emailaddress',
                label: 'Email',
                required: true
            }
        ]
    }
});
```



Figure 5: E-mail optimized keyboard in iOS

## Defining URL Fields

The Url field extends the TextField class. At run time, it functions almost identically to a text field with the following two key distinctions:

- Data is validated to ensure proper formatting
- The device's onscreen keyboard is changed in order to facilitate data entry

You can instantiate email fields explicitly either through the Ext.form.Url constructor or implicitly through an Ext.form.FormPanel configuration object as indicated below:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield',
                name: 'name',
                label: 'Full Name',
                required: true ,
                autoFocus: true
            },
            {
                xtype: 'urlfield',
                name:'homepage',
                label: 'Home Page',
                required: true
            }
        ]
    }
});
```

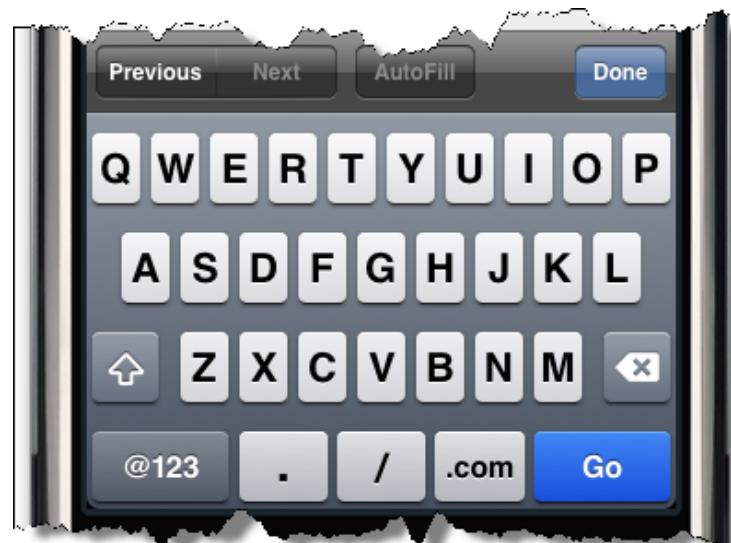


Figure 6: Optimized keyboard for URL fields in iOS

## Defining Text Areas

Textareas are represented in Sencha Touch through the Ext.form.TextArea class. You will typically instantiate them through your Ext.form.FormPanel constructor as demonstrated by the following code sample:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield', name: 'name',
                label: 'Full Name', required: true ,
                autoFocus: true
            },
            {
                xtype: 'textareafield',
                name: 'description',
                label: 'Description',
                maxRows: 3
            }
        ]
    ]
});
```

The following configuration property is specific to the Ext.form.TextArea class:

Config Option	Description
maxRows	The maximum number of lines made visible by the input



Figure 7: Text Area in iOS

## Defining Number Fields

Number fields are represented in Sencha Touch through the Ext.form.Number class. You will typically instantiate them through your Ext.form.FormPanel constructor as demonstrated by the following code sample:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield', name: 'name',
                label: 'Full Name', required: true,
                autoFocus: true
            },
            {
                xtype: 'numberfield',
                name: 'age',
                label: 'Age',
                required: true,
                minValue: 18,
                maxValue: 65,
                stepValue: 1
            }
        ]
    ]
});
```

Note that devices will typically display an optimized onscreen keyboard for numeric data entry. Some browsers may also represent number fields using a “spinner” UI.

The following configuration properties are specific to the Ext.form.Number class:



**Figure 8: Optimized keyboard for numeric entry in iOS**

Config Option	Description
minValue	Minimum value allowed
maxValue	Maximum value allowed
stepValue	Increment or decrement by number of units

## Deploying a Date Picker

DatePicker fields are represented in Sencha Touch through the Ext.form.DatePicker class. You will typically instantiate them through your Ext.form.FormPanel constructor as demonstrated by the following code sample:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            { xtype: 'textfield', name: 'name',
                label: 'Full Name', required: true},
            {
                xtype: 'datepickerfield',
                name: 'dob',
                label: 'Birth Date',
                required: true,
                picker: { yearFrom: 2009, yearTo: 2020 },
                value: { year: 2001, day: 11, month: 11 }
            }
        ]}
    ]
});
```

Ext.form.DatePickerField extends the Ext.form.Field element and supports the following class-specific configuration options:

Configuration Option	Description
datePickerConfig	Object. used when creating the internal Ext.DatePicker component.
value	Object/Date. Default value for the component



Figure 9: Date Picker

## Grouping Fields with a Fieldset

FieldSets contain fields as items. FieldSets do not add any behavior, other than an optional title, and are just used to group similar fields together. Wrapping fields in a fieldset does slightly alter their default appearance by adding rounded corners as depicted in the iOS screen shot below.

While Fieldsets can be instantiated through the Ext.form.FieldSet constructor, they are typically defined within a FormPanel as demonstrated by the following code snippet:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield',
                name: 'lname',
                label: 'Last Name',
                required: true,
                autoFocus: true
            },
            {
                xtype: 'textfield',
                name: 'fname',
                label: 'First Name',
                required: true
            }
        ]
    ]
});
```



Figure 10: A Fieldset in iOS

Fieldsets can use the following class-specific configuration options:

Config Option	Description
title	Optional title, rendered above grouped fields
instructions	Optional fieldset instructions, rendered below grouped fields

## Using Checkboxes

Checkbox fields are represented in Sencha Touch through the Ext.form.Checkbox class. It is typically defined as indicated below:



Figure 11: Sencha Touch checkbox in iOS

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'fieldset', title: 'About Me', items: [
            {
                xtype: 'textfield', name: 'name',
                label: 'Full Name', required: true
            },
            {
                xtype: 'checkboxfield',
                name: 'delivery',
                label: 'Request Delivery',
                checked: true,
                value: 1
            }
        ]
    ]
});
```

Ext.form.Checkbox extends Ext.form.Field. It also supports the following class-specific configuration attributes:

Config Option	Description
checked	Boolean. Sets the default checked state
value	The value of the field if checked.

## Defining Radio Buttons

Radio buttons are represented in Sencha Touch through the Ext.form.Radio class. It extends the Ext.form.Checkbox class and is typically defined as indicated below:

```
onReady: function() {
    mypanel = new Ext.form.FormPanel({
        fullscreen: true,
        items: [
            { xtype: 'fieldset', title: 'Favorite Game', items: [
                {
                    xtype: 'radiofield',
                    name: 'favGame',
                    label: 'World of Warcraft',
                    checked: true,
                    value: 1
                },
                {
                    xtype: 'radiofield',
                    name: 'favGame',
                    label: 'Star Trek Online',
                    value: 2
                },
                {
                    xtype: 'radiofield',
                    name: 'favGame',
                    label: 'Star Wars Old Republic',
                    value: 3
                }
            ]});
    });
}
```



Figure 12: Radio buttons in iOS

Note the following:

- A fieldset is used to visually group the radio buttons together
- Button selection is mutually exclusive since the name attribute of each button has an identical value

## Implementing a Select Box

The Select field (Ext.form.Select) extends Ext.form.Field and is typically used to allow a user to select a single item from a list of mutually exclusive values.

Ext.form.Select supports the following class-specific configuration options:

Config Option	Description
displayField	String/Integer. The underlying data value name (or numeric Array index) to bind to this Select control. This resolved value is the visibly rendered value of the available selection options. (defaults to 'text')
options	Array. An inline array of selection option objects containing corresponding valueField and displayField suitable for rendering the options list.
prependText	A static string to prepend before the active item's text when displayed as the select's text. Defaults to "".
store	Ext.data.Store. store instance used to provide selection options data.
valueField	The underlying data value name (or numeric Array index) to bind to this Select control. (defaults to 'value')

The following code snippet illustrates the syntax for deploying a Select box.

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        {
            xtype: 'textfield',
            name: 'fullname',
            label: 'Name'
        },
        {
            xtype: 'selectfield',
            name: 'candy',
            label: 'Favorite Candy',
            options: [
                {text: 'Skittles', value: 1},
                {text: 'Nerds', value: 2},
                {text: 'Bottle Caps', value: 3},
                {text: 'Sweet Tarts', value: 4}
            ],
            value: 3
        }
    ]
});
```



Figure 13: Select Box in iOS

```
    ]  
});
```

## Implementing a Slider Control

The Slider control (Ext.form.Slider) extends Ext.form.Field and allows a user to move a 'thumb' along a slider axis to choose a value. Note that sliders can be used outside the context of a form.



**Illustration 1:** A slider control

Ext.form.Slider class-specific configuration options include the following:

Config Option	Description
animate	Boolean. If true (default), the slider thumbs are animated when their values change.
increment	Number. The increment by which to snap each thumb when its value changes. Defaults to 1. Any thumb movement will be snapped to the nearest value that is a multiple of the increment (e.g. if increment is 10 and the user tries to move the thumb to 67, it will be snapped to 70 instead)
maxValue	Numeric. The highest value any thumb on this slider can be set to (defaults to 100)
minValue	Numeric. The lowest value any thumb on this slider can be set to (defaults to 0)
thumbs	Array. Optional array of Ext.form.Slider.Thumb instances. Usually values should be used instead.
value	The value to initialize the thumb at (defaults to 0)

Config Option	Description
values	Array. The values to initialize each thumb with. One thumb will be created for each value. This configuration should always be defined but if it is not then it will be treated as though [0] was passed.

The following example illustrates the syntax for deploying a slider control:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'textfield', name: 'name', label: 'Vendor' },
        { xtype: 'slider',
            name: 'quality',
            label: 'Quality',
            minValue: 1, maxValue: 10,
            increment: 1, value: 5
        }
    ]
});
```

## Implementing a Spinner Control

The Spinner control (Ext.form.Spinner) extends Ext.form.Field and wraps an HTML 5 number field.

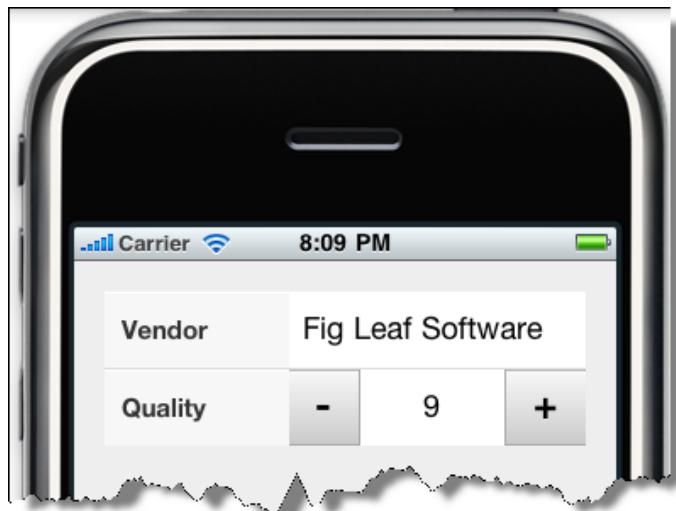


Illustration 2: Spinner Control

Ext.form.Spinner class-specific configuration options include the following:

Config Option	Description
accelerate	Boolean. True (default) if autorepeating should start slowly and accelerate.
cycle	Boolean. When set to true, if the maximum value is reached, the value will be set to the minimum. If the minimum value is reached, the value will be set to the maximum. Defaults to false.
defaultValue	Number. Initial value for the field
disableInput	Boolean. If set to true, the input field is disabled and only the spinner buttons may be pressed.
incrementValue	Number. Value that is added or subtracted from the current value when a spinner is used. Defaults to 1.
maxValue	Number. The maximum allowed value.
minValue	Number. The minimum allowed value.

The following example illustrates a sample deployment of the Spinner control:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'textfield', name: 'name', label: 'Vendor' },
        {
            xtype: 'spinnerfield',
            name: 'quality',
            label: 'Quality',
            minValue: 1,
            maxValue: 10,
            incrementValue: 1,
            value: 5,
            cycle: true
        }
    ]
});
```

## Implementing a Toggle Control

The Toggle control (Ext.form.Toggle) extends Ext.form.Slider. It is a specialized Slider with a single thumb and only two values. By default the toggle component can be switched between the values of 0 and 1.

The following configuration options are specific to Ext.form.Toggle:

Config Option	Description
maxValueCls	String. A CSS class added to the field when toggled to its maxValue
minValueCls	String. A CSS class added to the field when toggled to its minValue

You can either deploy a Toggle control through the Ext.form.Toggle constructor or implicitly through a form panel as illustrated by the following example:

```
mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'textfield', name: 'name', label: 'Vendor' },
        { xtype: 'togglefield',
            name: 'active', label: 'Active?' }
    ]
});
```



Figure 14: Toggle Control in iOS

## Defining Hidden Fields

While hidden fields may be instantiated through the Ext.form.Hidden constructor, they are typically defined through the configuration options object of Ext.form.FormPanel. Ext.form.Hidden extends the Ext.form.Field class. The following example illustrates defining a hidden field in a form:

```
var formPnl = new Ext.form.FormPanel({  
    items: [  
        {  
            xtype: 'hiddenfield',  
            name: 'datarecordprimarykey',  
            value: 1  
        }  
    ]  
});
```

## Walkthrough 4-1: Implementing a Form Design



In this walkthrough, you will define a form to report a crime:

- Instantiate the form using the Ext.form.FormPanel constructor
- Define form fields
- Add a toolbar with a submit button

Note: You will add code to submit the form later in this unit.

### Steps

#### Define the Form

1. Open **/walk/walk4-1/index.html** in your editor
2. Where indicated by the comment, modify the constructor to generate a **FormPanel**. Your code should appear as follows:

```
var reportPnl = new Ext.form.FormPanel ({ });
```

3. Add a configuration property to the **FormPanel** in order to enable vertical scrolling.
4. Add an **items** array to the **FormPanel** configuration object. Your code should appear similar to the following:

```
var reportPnl = new Ext.form.FormPanel ({  
    scroll: 'vertical',  
    items: [  
        ]  
});
```

5. In the **items** array, define a **FieldSet** with the following attributes:
  - title: 'About me'
6. In the **fieldset**, define the following input fields with label widths of 110 pixels:

xtype	name	label	required
textfield	lname	Last Name	TRUE
textfield	fname	First Name	TRUE
emailfield	Email	E-mail	TRUE

Your items array should appear similar to the following:

```
items: [
  {
    xtype: 'fieldset', title: 'About me', items: [
      {
        xtype: 'textfield',
        name: 'lname',
        label: 'Last Name',
        labelWidth: 110,
        required: true
      },
      {
        xtype: 'textfield',
        name: 'fname',
        label: 'First Name',
        labelWidth: 110,
        required: true
      },
      {
        xtype: 'emailfield',
        name: 'email',
        label: 'E-mail',
        labelWidth: 110,
        required: true
      }
    ]
  }
]
```

7. Insert a telephone field directly underneath the emailfield.
8. Save the file and test

#### Add a Select control

9. Return to your editor
10. Add another **fieldset** to your form with the following attribute:
  - title: 'Offense'
11. Inside the Offense **fieldset**, add a **SELECT** control containing the following options:

Text	Value
Jaywalking	1
Littering	2
Speeding	3

12. Save the file and test.

Your code should appear similar to the following:

```
{  
    xtype: 'selectfield',  
    label: 'Violation',  
    options: [  
        {text: 'Jaywalking', value: '1'},  
        {text: 'Littering', value: '2'},  
        {text: 'Speeding', value: '3'}  
    ]  
}
```

### Add a Date Picker

13. Underneath the select control that you added in step 11, insert a Date Picker with the following attributes:

- name: offendedsdate
- label: Date
- required: true
- value: new Date()

Your code should appear similar to the following:

```
{  
    xtype: 'datepickerfield',  
    name: 'offendedsdate',  
    label: 'Date',  
    required: true,  
    value: new Date(),  
    picker: { yearFrom: 2009, yearTo: 2020 },  
}
```

14. Save the file and test

### Add a Checkbox

15. Add a **checkbox** underneath the date picker that you inserted in step 12. Use the following attributes;

- name: currentLocation
- label: Current Location
- value: 1
- checked: true

Your code should appear similar to the following:

```
{  
    xtype: 'checkboxfield',  
    name: 'currentLocation',  
    label: 'Current Location',  
    inputValue: 1,  
    checked: true  
}
```

### Add a Text Field

16. Add a text field underneath the checkbox that you defined in step 15.  
Use the following attributes:
  - name: address
  - label: Address
17. Save the file and test
18. Return to your editor

### Add a Spinner Field

19. Add a spinner field underneath the text field that you defined in step 16. Use the following attributes:
  - name: speed
  - label: Top Speed
  - minValue: 25
  - maxValue: 125
  - incrementValue: 5
  - value: 25
  - cycle: true
20. Save the file and test

### Add a Text Area

21. Add a textarea field underneath the spinner field that you defined in step 19. Use the following attributes:
  - name: description
  - label: description
  - labelAlign : top
  - labelWidth: 100%
22. Save the file and test

**Add a Docked Toolbar with a Submit Button**

23. Add a **toolbar**, docked to the bottom of the form, with a single button labeled ‘Prepare to be Judged’.
24. Center the button on the toolbar. Your code should appear similar to the following:

```
dockedItems: [
  {
    dock: 'bottom',
    xtype: 'toolbar',
    layout: {pack: 'center'},
    items: [
      {text: 'Prepare to be Judged'}
    ]
  }
]
```

25. Save the file and test

– End of Walkthrough --

## Handling Form Events



Typically you will want to design forms that are context-sensitive, involving field-level dependencies. For example, it only makes sense for the “confess” form that you developed during the last walkthrough to display its “Top Speed” field if you chose “Speeding” as your offense.

In order to create forms that hide and show fields based on user interaction you will need to create event handlers and invoke public methods to retrieve field values and toggle field visibility.

## Handling Changes in Field Values

All of the form element classes covered previously in this unit inherit from the Ext.form.Field class. This class raises the following events:

- blur (Ext.form.Field this)  
Fires when the field loses input focus
- change ( Ext.form.Field this, Mixed newValue, Mixed oldValue)  
Fires just before the field blurs if the field value has changed
- keyup (Ext.form.Field this, Ext.EventObject e)  
Fires when a key is released on the input element

In addition, the following events are component-specific:

Component	Event	Description
Ext.form.Select	select	Triggered when the user makes a selection
Ext.form.Checkbox	check	Triggered when the user checks the checkbox.
Ext.form.Checkbox	uncheck	Triggered when the user unchecks the checkbox

You can add an event handler to your form fields using the listeners attribute of your configuration object as indicated by the following code snippet:

```
{  
    xtype: 'textfield',  
    name: 'title',  
    label: 'Title',  
    required: true,  
    listeners: {change: onTitleChange}  
}
```

Typically your event handlers evaluate the value of the form field and then either change the values or hide/show dependent fields. All form fields support the following methods:

Method	Description
getValue()	Returns a field's normalized data value
setValue()	Sets a data value into the field and validates it
hide()	Hides the component
show()	Show the component
setDisabled()	Enable or disable the component
reset()	Resets the current field value to its original value
focus()	Attempts to set the field as the active input focus

Use the `Ext.getCmp (String id)` method to set a pointer to a component.

The following example illustrates defining a change handler for a text field that, in turn, copies its value into a second text field:

```

onTitleChange = function(obj, newVal, oldVal) {
    var destObj = Ext.getCmp('description');
    if (destObj.getValue() == '')
        destObj.setValue(newVal);
}

mypanel = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        {
            xtype: 'textfield',
            name: 'title',
            label: 'Title',
            id: 'title',
            listeners: {change: onTitleChange}
        },
        {
            xtype: 'textfield',
            name: 'description',
            label: 'Description',
            id: 'description'
        }
    ]
});

```

## Walkthrough 4-2: Implementing Field Dependencies



During this exercise you will modify the crime reporting form that you created in the prior walkthrough to make some of the fields context-sensitive:

- Define an event handler
- Change the visibility of form fields



Figure 15: The Top Speed field only appears if Speeding is selected as the offense

### Steps

#### Review the form

1. Open [/walk/walk4-2/index.html](#) in your editor
2. Review the contents with your instructor. Note that ID attributes have been added to all the form fields.

### Add context sensitivity to the Address field

3. Add a **hidden** attribute to the **address** textfield that initially hides it from view
4. Add a listener attribute to the **currentLocation** checkbox that invokes a function named **currentLocationCheck** whenever the user checks or unchecks the field. Your full checkbox definition should resemble the following:

```
{
    xtype: 'checkboxfield',
    name: 'currentLocation',
    label: 'Current Location',
    inputValue: 1,
    checked: true,
    listeners: {
        check: currentLocationCheck,
        uncheck: currentLocationCheck
    }
}
```

5. Where indicated by the comment, define a function named **currentLocationCheck** that accepts an argument named **obj**. Your code should appear similar to the following:

```
var currentLocationCheck = function(obj) { }
```

6. Inside the **currentLocationCheck()** function, insert an **IF** block that evaluates the value of **obj.isChecked()**
  - If the value is **true**, hide the **address** component.
  - If the value is **false**, show the **address** component.

Your code should appear similar to the following:

```
var currentLocationCheck = function(obj) {
    if (obj.isChecked()) {
        Ext.getCmp('address').hide();
    } else {
        Ext.getCmp('address').show();
    }
}
```

7. Save the file and test. Checking and unchecking the Current Location checkbox should toggle the visibility of the Address field.

### Add context sensitivity to the Select box

8. Return to your editor
9. Add a **hidden** attribute to the **speed** spinnerfield that initially hides it from view

10. Add a listener attribute to the **offense** select box that invokes a function named **offenseChange** whenever the user changes their selection.  
Your full select box definition should resemble the following:

```
{  
    xtype: 'selectfield',  
    name: 'offense',  
    options: [  
        {text: 'Jaywalking', value: 1},  
        {text: 'Littering', value: 2},  
        {text: 'Speeding', value: 3}  
    ],  
    listeners: {change: offenseChange}  
}
```

11. Where indicated by the comment, define a function named **offenseChange** that accepts three arguments – **obj**, **newValue**, and **oldValue**

Your code should resemble the following:

```
offenseChange = function(obj, newValue, oldValue) {}
```

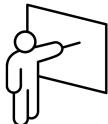
12. Inside the offenseChange function, insert an IF statement that evaluates **newValue**.
  - If **newValue** equals **3** then show the speed field
  - If **newValue** is not **3** then hide the speed field
13. Your code should appear similar to the following:

```
var offenseChange = function(obj,newValue,oldValue) {  
    if (obj.getValue() == 3) {  
        Ext.getCmp('speed').show();  
    } else {  
        Ext.getCmp('speed').hide();  
    }  
}
```

14. Save the file and test. Changing the value in the select box should toggle the visibility of the speed field.

– End of Walkthrough --

## Loading Data from a Model



A Model represents a data object that your application manages. For example, you might define a Model for Users, Products, Cars, or any other real-world object that you need to model in the system. Models are registered via the model manager and are used by stores, which are in turn used by many of the data-bound components in Ext.

Models are defined as a set of fields and any arbitrary methods and properties relevant to the model. Data instances of a model may be used to populate a form as indicated by the following example:

```
myForm = new Ext.form.FormPanel({
    fullscreen: true,
    items: [
        { xtype: 'textfield',
            name: 'firstname',
            label: 'First Name',
            id: 'firstname'
        },
        { xtype: 'textfield',
            name: 'lastname',
            label: 'Last Name',
            id: 'lastname'
        },
        {
            xtype: 'numberfield',
            name: 'age',
            label: 'Age',
            id: 'age'
        },
        {
            xtype: 'textfield',
            name: 'ssn',
            label: 'SSN',
            id: 'ssn'
        }
    ]
});

Ext.regModel('User', {
    fields: [
        {name: 'firstname', type: 'string'},
        {name: 'lastname', type: 'string'},
        {name: 'age', type: 'int'},
        {name: 'ssn', type: 'string'}
    ]
});
// create new model instance
var person = Ext.ModelMgr.create({
    firstname: 'Steve',
    lastname : 'Drucker',
    age      : 41,
    ssn     : '555-55-5555'
}, 'User');

myForm.load(person); //load data into form
```

## Defining a Model

Define your models through the Ext.regModel() constructor. Typically models consist of the following:

- idProperty to identify the primary key field
- Field definitions
- Custom Methods
- Validation Rules

### Defining fields in a Model

You define fields in a model through the fields attribute of the Ext.regModel() constructor. Typically the names of your fields should match with your form field names. The following example defines a simple model:

```
Ext.regModel('User', {
    idProperty: 'userid',
    fields: [
        {name: 'userid', type: 'int'},
        {name: 'firstname', type: 'string'},
        {name: 'lastname', type: 'string'},
        {name: 'age', type: 'int', defaultValue: 20},
        {name: 'ssn', type: 'string'}
    ]
});
```

Fields definitions include the following properties:

Config Option	Description
allowBlank	Boolean. Used for validating a record. Defaults to true.
convert	Function. A function that transforms the value.
dateFormat	String. Used when converting received data into a Date
defaultValue	Mixed. The default value used when a record is created. Defaults to the empty string.
name	String. The name by which the field is referenced within the record.
sortDir	Initial direction to sort. Valid values are “ASC” (default) and “DESC”
sortType	Function. A function that converts a field value to ensure correct sort ordering.

Config Option	Description
type	Mixed. The data type for automatic conversion from received data to the stored value if a convert function has not been defined. Possible values are: auto (default), string, int, float, boolean, date
useNull	Boolean. Use when converting received data into a Number type (either int or float). If the value cannot be parsed, null will be used if useNull is true, otherwise the value will be 0. Defaults to false.

The following example illustrates using many of the configuration options described above:

```
function fullName(v, record) {
    return record.lastname + ', ' + record.firstname;
}

Ext.regModel('User', {
    fields: [
        {name: 'firstname', type: 'string', allowBlank: false},
        {name: 'lastname', type: 'string', allowBlank: false},
        {name: 'fullname', type: 'string', convert: fullName},
        {name: 'age', type: 'int', useNull: true},
        {name: 'ssn', type: 'string', allowBlank: false}
    ]
});
```

## Defining Custom Methods in a Model

Models support a number of public methods including the following:

Method	Description
get(String fieldname)	Returns the value of the field
set(String fieldname, Mixed value)	Sets the value of the field

Note that you can also define custom methods within the constructor as indicated by the following example:

```
Ext.regModel('User', {
    fields: [
        {name: 'firstname', type: 'string', allowBlank: false},
        {name: 'lastname', type: 'string', allowBlank: false},
        {name: 'fullname', type: 'string', convert: fullName},
        {name: 'age', type: 'int', useNull: true},
        {name: 'ssn', type: 'string', allowBlank: false}
    ],

    changeName: function() {
        var newName = "Darth " + this.get('firstname');
        this.set("firstname",newName);
    }
});
```

## Defining Validation Rules

Configure your validation rules by passing parameters into the Ext.data.validations class. The following five validation rules are built-in to Sencha Touch:

Rule	Description
presence	Ensures that a field has a value. Zero is valid, however, the empty string is not.
length	Ensures that a string is between a min and max length. Both constraints are optional
format	Ensures that a string matches a regular expression format.
inclusion	Ensures that a value is within a specific set of values
exclusion	Ensures that a value is not one of the specific set of values

The following example illustrates defining validations on a model:

```
Ext.regModel('User', {
    idProperty: 'userid',
    fields: [
        {name: 'userid', type: 'int'},
        {name: 'firstname', type: 'string', allowBlank: false},
        {name: 'lastname', type: 'string', allowBlank: false},
        {name: 'fullname', type: 'string', convert: changeName},
        {name: 'age', type: 'int', useNull: true},
        {name: 'ssn', type: 'string', allowBlank: false},
        {name: 'gender', type: 'string', allowBlank: false}
    ],

    changeName: function() {
        var newName = "Darth " + this.get('firstname');
        this.set("firstname",newName);
    }
});
```

```
validations: [
    {type: 'presence',
     name: 'firstname',
     message: 'Please enter your First Name'
    },
    {type: 'presence',
     name: 'lastname',
     message: 'Please enter your last name'
    },
    {
        type: 'format',
        name: 'ssn',
        matcher: /^[0-9]{3}[-]?[0-9]{2}[-]?[0-9]{4}$/,
        message: 'Your email must be in the format u@me.com'
    },
    {
        type: 'inclusion',
        name: 'gender',
        list: ['M','F']
        message: 'Valid options are "M" or "F" for gender'
    }
]
});
```

## Creating a Data Instance from a Model

Create instances of your model using the Ext.ModelMgr class using its create method as indicated by the following example:

```
var user = Ext.ModelMgr.create({
    firstname: 'Steve',
    lastname : 'Drucker',
    age      : 41,
    ssn      : '555-55-5555' ,
    gender: 'M'
}, 'User');
```

As indicated by the prior example, the first parameter is a configuration object that maps to the fields defined in your model. The second parameter is the name of the model that you had defined previously.

## Binding a Model Instance to a Form

Instances of models can be bound to a form using the Ext.form.FormPanel load method as indicated below:

```
myForm.load(user);
```

Conversely, you can use the Ext.form.FormPanel updateRecord() method to transfer data from a form back into your model instance as indicated below:

```
myForm.updateRecord(user, true);
```

## Evaluating Validation Rules

When a user presses a submit button you will typically load the data from the form back into your model instance and then evaluate the data in your model to verify whether it passes all your defined validation rules. If any of the validation rules fail you will need to present the user with a message explaining what they need to fix.

## Executing Validation Checks

You can trigger the validation rules on a model by invoking the instance's validate() method which returns an Ext.data.Errors object as indicated below:

```
submitForm = function() {
    myForm.updateRecord(user, true);
    var errors = user.validate();
}
```

Public methods for Ext.data.Errors include the following:

- **isValid()**  
Returns true if there are no errors in the collection
- **forField (string fieldname)**  
Returns all errors for the given field
- **each (function)**  
Executes a function for each error returned

Note that each error item contains the following properties:

Property	Description
field	The name of the field that failed validation
message	A description of the failure

The following example uses the methods and properties to aggregate error messages into a single string:

```
submitForm = function() {
    var errorstring = "";
    myForm.load(user);
    errors = user.validate();
    if (!errors.isValid) {
        errors.each(function (errorObj){
            errorstring += errorObj.field + ":" +
                errorObj.message + "<br>";
        })
    }
}
```

## Outputting Error Messages in an Alert Box

The Ext.MessageBox class contains a number of methods that enable you to output modal popup boxes. Note that unlike desktop browser-based invocations, these Sencha Touch methods execute asynchronously. You can invoke each of the following methods from the Ext.Msg singleton class:

- **alert(String title, String msg, [Function fn], [Object scope])**  
Displays a standard read-only message box with an OK button (comparable to the basic JavaScript alert prompt). If a callback function is passed it will be called after the user clicks the button, and the itemId of the button that was clicked will be passed as the only parameter to the callback.
- **confirm (String title, String msg, [Function fn], [Object scope])**  
Displays a confirmation message box with Yes and No buttons (comparable to JavaScript's confirm). If a callback function is passed it will be called after the user clicks either button, and the id of the button that was clicked will be passed as the only parameter to the callback (could also be the top-right close button).

- **prompt (String title, String msg, [Function fn], [Object scope], [Boolean/number multiline],[String value], Object promptConfig)**

Displays a message box with OK and Cancel buttons prompting the user to enter some text. The prompt can be either a single line or a multi-line textbox. If a callback function is passed, If a callback function is passed it will be called after the user clicks either button, and the id of the button that was clicked (could also be the top-right close button) and the text that was entered will be passed as the two parameters to the callback.

The syntax for generating a modal alert popup box is the following:

```
submitForm = function() {  
    var errorstring = "";  
    myForm.updateRecord(user);  
    errors = user.validate();  
    if (!errors.isValid) {  
        errors.each(function (errorObj){  
            errorstring += errorObj.field + ":" +  
                errorObj.message + "<br>";  
        }  
        Ext.Msg.alert('Errors in your input',errorstring);  
    }  
}
```

## Walkthrough 4-3: Adding Form Validation



In this walkthrough, you will modify your form to include input validation.

- Define a data model
- Populate a form from a data model
- Populate a data model from a form
- Apply validation rules
- Report validation errors

### Steps

#### Review the form

1. Open [/walk/walk4-3/index.html](#) in your editor. This is the solution from the last walkthrough.

#### Define a Data Model

2. Where indicated by the comment, define a data model named '**'CrimeReport'**' that represents the information displayed in the form. Your code should appear similar to the following:

```
Ext.regModel ('CrimeReport',
{
    fields: [
        {name: 'lname', type: 'string', allowBlank: false },
        {name: 'fname', type: 'string', allowBlank: false },
        {name: 'email', type: 'string', allowBlank: false},
        {name: 'offense', type: 'int', allowBlank: false },
        {name: 'offensedate', type: 'date', allowBlank: false},
        {name: 'currentLocation', type: 'int'},
        {name: 'address', type: 'string'},
        {name: 'speed', type: 'int'}
    ]
});
```

**Define validation rules for the data model**

3. Inside the `Ext.regModel()` invocation that you created in the prior step, define the following validations:

Field	Type	Matcher
fname	presence	Not applicable
lname	presence	Not applicable
email	format	/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\$/

Your validators should appear similar to the following:

```
validations: [
  {
    type: 'presence',
    name: 'fname',
    message: 'Enter your first name'
  },
  {
    type: 'presence',
    name: 'lname',
    message: 'Enter your last name'
  },
  {
    type: 'format',
    name: 'email',
    message: 'Your email address is an invalid format',
    matcher: /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\$/}
]
```

**Define an instance of the model**

4. Where indicated by the comment, create a new instance of the **CrimeReport** model named **crimeReportData**.
- Set the default values for all **string** variables as the **empty string**.
  - Set **currentLocation** to 1
  - Set the **speed** to 25

Your code should appear similar to the following:

```
var crimeReportData = Ext.ModelMgr.create({
  lname: '',
  fname: '',
  email: '',
  offense: 1,
  offendedsdate: '',
  currentLocation: 1,
  address: '',
  speed: 25},
  'CrimeReport'
);
```

5. Immediately after the code that you inserted from the prior step, load the **crimeReportData** into your form as indicated below:

```
reportPnl.load(crimeReportData);
```

### Validate the Data and Output a Response

6. Where indicated by the comment, define a handler function for the submit button. Your code should appear as follows:

```
handler: function() { }
```

7. Inside the handler function that you defined in the previous step, define a local variable named **errorString** and initialize it as an empty string.
8. Post the form data back into your data model instance as indicated below:

```
reportPnl.updateRecord(crimeReportData);
```

9. Validate the data in your data model instance, placing the results into a local variable named errors.

```
var errors = crimeReportData.validate();
```

10. Under the code that you inserted in the prior step, insert an IF block to check if any errors were reported:

```
if (!errors.isValid()) {
} else {
}
```

11. If validation errors were detected, loop through them, outputting the field name and message into the errorstring variable as indicated below:

```
errors.each(function (errorObj) {
    errorString += errorObj.message;
    errorString += "<br>";
})
```

12. Immediately after the code that you inserted in the prior step, output the contents of the errorstring variable into an alert box:

```
Ext.Msg.alert('Errors in your input',errorstring);
```

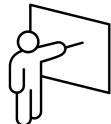
13. Inside the else section of the if block that you inserted from step 11, output the text string 'OK' to the console log:

```
console.log ('ok');
```

14. Save the file and test.

– End of Walkthrough --

## Submitting Form Data to a Server



You can submit form data to a server using the Ext.form.FormPanel's submit() method. The submit() method enables you to specify a configuration object as a parameter with the following attributes:

Attribute	Description
url	String. The url for the action
method	String. The form method to use (defaults to POST)
params	String/Object. The params to pass (defaults to the Form Panel's baseParams or none if not defined). Parameters are encoded as standard HTTP parameters using Ext.urlEncode.
headers	Object. Request headers to set for the action (defaults to the form's default headers)
autoAbort	Boolean. Set to true to abort any pending AJAX request prior to submission (defaults to false). Has no effect when standardSubmit is enabled.
submitDisabled	Boolean. Set to true to submit all fields regardless of disabled state (defaults to false). Has no effect when standardSubmit is enabled.
waitMsg	String/Config. If specified, the value is applied to the waitTpl if defined, and rendered to the waitMsgTarget prior to a Form submit action.
success	Function. The callback that will be invoked after a successful response. The function is passed the Ext.FormPanel that requested the action and the result object returned by the server as a result of the submit request.
failure	Function. The callback that is invoked after a failed transaction attempt. Passed the Ext.FormPanel that requested the submit and the result object returned from the server.
scope	Object. The scope in which to call the callback functions.

## Using the submit() method

The following example illustrates the syntax for performing a form submission and subsequently resetting the form fields back to their initial state:

```
myForm.submit({
    url: 'savemydata.php',
    method: 'post',
    submitDisabled: true,
    waitMsg: 'Saving Data...Please wait.',
    success: function (objForm,httpRequest) {
        Ext.Msg.alert("Record Saved");
        myForm.reset();
    }
})
```

## Coding the Action Page

Note that the form submission process occurs in the background using an AJAX connection to the server. In this circumstance, Sencha Touch expects your action page to return a result in JSON format.

The following example illustrates returning a simple success code using a PHP-based action page:

```
<?php
header('Content-Type: application/json');
echo '{"success":true, "msg":'.json_encode('Thank you for
your submission').'}';
?>
```

A similar action page in ColdFusion would resemble the following:

```
<cfcontent type="application/json">{"success": true,
"msg": "Thanks"}
```

## Walkthrough 4-4: Posting Form Data



In this walkthrough, you will add code to your form's submit button in order to send the data to your application server.

- Define a submit action
- Output a success message

### Steps

#### Review the form

1. Open [/walk/walk4-4/index.html](#) in your editor. This is the solution from the last walkthrough.

#### Define a Submit Action

2. Where indicated by the comment, invoke the `submit()` method of your form, passing in the following configuration properties:
  - `url:` 'savereport.json'
  - `method:` 'post'
  - `submitDisabled:` true
  - `waitMsg:` 'Saving Data...Please Wait to Be Judged"
3. Add a **success handler** to the configuration object that you defined in the prior step that performs the following actions:
  - Output an alert box stating the record has been saved
  - Reset the form fields back to their defaults
4. Verify that your code resembles the following:

```
reportPnl.submit({  
    url: 'savereport.json',  
    method: 'post',  
    submitDisabled: true,  
    waitMsg: 'Saving Data...Please wait.',  
    success: function (objForm,httpRequest) {  
        Ext.Msg.alert("Record Saved");  
        reportPnl.reset();  
    }  
})
```

5. Save the file and test
6. Open **savereport.json** and review its contents with your instructor.

– End of Walkthrough --

## Unit Summary



- The Ext.form.FormPanel class acts as the foundation of your forms
- Sencha Touch supports the following form elements:
  - Check Box
  - Date Picker
  - Email Field
  - Fieldset
  - Hidden
  - Number
  - Password
  - Radio Buttons
  - Search
  - Select
  - Slider
  - Spinner
  - Text
  - Text Area
  - Toggle
  - URL
- You can attach listeners to form field events in order to create context dependencies between the input components.
- Define data models and associated validation rules to load information into your form and perform validation checks.
- Use the Ext.form.FormPanel submit() method to transmit data from your form to the server via AJAX
- Sencha Touch AJAX submissions expect a JSON response

## Unit Review



1. List three configuration attributes that are common to all form fields.
2. What are the differences between an email field and a text field?
3. Describe a use-case for a slider control.
4. List three form field events that can be used to trigger code when a user modifies data.
5. Why is it important to define a data model?
6. Data must be output from your form action page in what format?

# Lab 4: Applying for Admission to SubGenius University



During this lab you create the admissions form, depicted below, for the fictional SubGenius University.

The screenshot shows a web-based application for university admissions. At the top, a dark header bar contains the "SubGenius University" logo and four navigation links: "Welcome", "Courses", "Schedule", and "Enroll". Below the header, the main content area has a light gray background. It features two sections: "About me" and "Academic Information". The "About me" section contains three input fields for "Last Name", "First Name", and "E-mail". The "Academic Information" section includes a checkbox for "H.S. Graduate?", which is checked. It also features a numeric input field for "High School GPA" with a value of "2.74", accompanied by minus and plus buttons for adjustment. A dropdown menu for "Proposed Major" is set to "Please Select". At the bottom of the form is a dark footer bar with a "Submit Application" button.

## Objectives

After completing this lab, you should be able to:

- Define the application form
- Add validation rules
- Submit the form data to a server

## Steps

### Define the Form Panel

1. Open **/lab4/index.htm** and review its contents
2. Where indicated by the comment, change the **enrollPnl** definition from a **Panel** to a **FormPanel** with the following attributes:
  - **scroll:** vertical
  - **items:** [ ]
  - **dockedItems:** [ ]

### Define a Fieldset for contact information

3. Define a **fieldset** as the first item in the items array with the following attributes:
  - **xtype:** fieldset
  - **title:** 'About Me'
  - **items:** [ ]
4. In the items array for the fieldset, define the following fields:

<b>xtype</b>	<b>name</b>	<b>label</b>	<b>required</b>
textfield	lname	Last Name	TRUE
textfield	fname	First Name	TRUE
emailfield	email	E-mail	TRUE

5. Save the file and browse
6. Verify that your code appears similar to the following:

```
var enrollPnl = new Ext.form.FormPanel({  
    scroll: 'vertical',  
    items: [  
        {xtype: 'fieldset', title: 'About me', items: [  
            {xtype: 'textfield', name: 'lname',  
                label: 'Last Name', required: true},  
            {xtype: 'textfield', name: 'fname',  
                label: 'First Name', required: true},  
            {xtype: 'emailfield', name: 'email',  
                label: 'E-mail', required: true }  
        ]},  
        dockedItems: []  
    ]};
```

### Define a Fieldset for the Academic Information

7. Add a second fieldset to the form with the following attributes:
  - **xtype:** 'fieldset'
  - **title:** Academic Information
  - **items:** [ ]
8. Add a checkbox to the Academic Information fieldset with the following attributes:
  - **xtype:** 'checkboxfield'
  - **name:** 'hsdiploma'
  - **label:** 'H.S. Grad?'
  - **value:** 1
9. Add a **spinnerfield** to the Academic Information fieldset with the following attributes:
  - **id:** 'gpa'
  - **name:** 'gpa'
  - **label:** 'H.S. GPA'
  - **minValue:** 0
  - **maxValue:** 4.4
  - **incrementValue:** 0.1
  - **value:** 2.0
  - **cycle:** true
10. Add an event listener to the spinnerfield that truncates the value to two decimal places:
 

```
listeners: {
  spin: function(obj, val) {
    obj.setValue(Ext.util.Numerics.toFixed(val, 2));
  }
}
```

11. Add a number field to the Academic Information fieldset with the following attributes:

- **xtype:** 'numberfield'
- **name:** 'greverbal'
- **id:** 'greverbal'
- **label:** 'GRE Verbal Score'
- **minValue:** 200
- **maxValue:** 800
- **stepValue:** 10
- **hidden:** true

12. Add a select box to the Academic Information fieldset with the following attributes:

- xtype: 'selectfield'
- name: 'major'
- label: 'Proposed Major'
- options: []

13. Add the following options to the select box:

Text	Value
Please Select	0
Web Development	1
Computer Graphics	2
Game Development	3

14. Save the file and browse.

15. Verify that your Academic Information fieldset resembles the following:

```
{
  xtype: 'fieldset',
  title: 'Academic Information',
  items: [
    {
      xtype: 'checkboxfield',
      name: 'hsdiploma',
      label: 'H.S. Graduate?',
      inputValue: 1,
    },
    {
      xtype: 'spinnerfield',
      id: 'gpa',
      name: 'gpa',
      label: 'H.S. GPA',
      minValue: 0,
      maxValue: 4.4,
      incrementValue: 0.1,
      value: 2.0,
      cycle: true,
      listeners: {
        spin: function(obj, val) {
          obj.setValue(Ext.util.Numerics.toFixed(val, 2));
        }
      }
    },
    {
      xtype: 'numberfield',
      name: 'greverbal',
      id: 'greverbal',
      label: 'GRE Verbal Score',
      minValue: 200,
      maxValue: 800,
      stepValue: 10,
      hidden: true
    }
  ]
}
```

```

},
{
    xtype: 'selectfield',
    name: 'major',
    label: 'Proposed Major',
    options: [
        {text: 'Please Select', value: 0},
        {text: 'Web Development', value: 1},
        {text: 'Computer Graphics', value: 2},
        {text: 'Game Development', value: 3}
    ]
}
]
}
]
}

```

### Create a Field Dependency

16. Add an event listener for the hsdiploma checkbox that invokes a function named **hsDiplomaHandler** whenever the checkbox is clicked.
17. Where indicated by the comment, define a function named **hsDiplomaHandler**
  - If the checkbox is checked, hide the **greverbal** field and show the **gpa** field
  - If the checkbox is not checked, hide the **gpa** field and show the **greverbal** field
18. Save the file and test
19. Verify that your hsdiploma checkbox resembles the following:

```

{
    xtype: 'checkboxfield',
    name: 'hsdiploma',
    label: 'H.S. Graduate?',
    inputValue: 1,
    listeners: {
        check: hsDiplomaHandler,
        uncheck: hsDiplomaHandler
    }
}

```

20. Verify that your event handler resembles the following:

```

hsDiplomaHandler = function(obj) {
    if (obj.isChecked()) {
        Ext.getCmp('gpa').show();
        Ext.getCmp('greverbal').hide();
    } else {
        Ext.getCmp('gpa').hide();
        Ext.getCmp('greverbal').show();
    }
}

```

### Define a Data Model and Associated Validation Rules

21. Define a data model named 'StudentApplication' that mimics the structure of your form fields. Configure all fields to be required except for 'greverbal'
22. Define validations for the following fields:

Field	Type	list	Matcher
fname	presence		
lname	presence		
email	format		/^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}\$/
major	exclusion	['0']	

23. Review your data model code to see if it resembles the following:

```
Ext.regModel ('StudentApplication',
{
    fields: [
        {name: 'lname', type: 'string', allowBlank: false },
        {name: 'fname', type: 'string', allowBlank: false },
        {name: 'email', type: 'string', allowBlank: false},
        {name: 'hsdiploma', type: 'int', allowBlank: false },
        {name: 'gpa', type: 'float', allowBlank: false},
        {name: 'greverbal', type: 'int'},
        {name: 'major', type: 'int'}
    ],
    validations: [
        {type: 'presence', field: 'fname'},
        {type: 'presence', field: 'lname'},
        {type: 'format', field: 'email', matcher: /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/},
        {type: 'exclusion', field: 'major', list: ['0']}
    ]
});
```

### Define a Data Instance and Apply it to the Form

24. Define an instance of the **StudentApplication** data model named **applicantData**. Initialize all string values to the empty string. Set the numeric values to the following:

- hsdiploma : 1
- gpa: 2.74
- greverbal: 400
- major: 0

25. Load the **applicantData** into your form.

26. Save the file and test.

27. Review your code to ensure it resembles the following:

```
var applicantData = Ext.ModelMgr.create(  
{  
    lname: '',  
    fname: '',  
    email: '',  
    hsdiploma: 1,  
    gpa: 2.74,  
    greverbal: '400',  
    major: 0  
, 'StudentApplication');  
  
enrollPnl.load(applicantData);
```

### Validate the Data and Submit the Form

28. Define a docked toolbar located at the bottom of the form panel that contains a button labeled “Submit Application”
29. Add a handler to the button that you created in the prior step.
30. Inside the handler, define a local variable named **errorstring** and initialize it to the empty string.
31. After the code that you inserted in the prior step, transfer the values from your form fields back into the data model instance.
32. Validate the data in your data model instance, placing the results into a variable named errors.
33. Insert an IF statement.
  - If validation errors exist, loop through them, adding appropriate messages onto the errorstring variable. Then display the errorstring variable in an alert box.
  - If there are no validation errors, submit the form to the url 'saveform.json', pop up an alert box indicated the application was submitted, and reset the form fields to their initial values.

34. Verify that your handler code resembles the following:

```
handler: function() {
    var errorstring = "";
    enrollPnl.updateRecord(applicantData);
    var errors = applicantData.validate();
    if (!errors.isValid()) {
        errors.each(function (errorObj){
            errorstring += errorObj.field + ":" +
errorObj.message + "<br>";
        })
        Ext.Msg.alert('Errors in your input',errorstring);
    } else {
        enrollPnl.submit({
            url: 'saveform.json',
            method: 'post',
            submitDisabled: true,
            waitMsg: 'Saving Data...Please wait.',
            success: function (objForm,httpRequest) {
                Ext.Msg.alert("Application Submitted");
                enrollPnl.reset();
            }
        })
    }
}
```

35. Save the file and test

– End of Lab --

---

---

# **Unit 5:**

## **Working with Structured Data**

### **Unit Objectives**

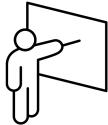
After completing this unit, you should be able to:

- Define data templates and use a DataView to format structured content
- Load data into a Model using AJAX
- Output data into a view
- Output a complex data structure into a nested list

### **Unit Topics**

- Using Data Templates to Generate HTML
- Reading Data from a Server into a Model
- Displaying Data in a List
- Displaying Hierarchical Data in a Nested List

## Using Data Templates to Generate HTML



Container objects can format structured data through a template. Use the `data` attribute of the configuration option to specify the name of a JavaScript array. Then define an EXT template element through the `tpl` attribute as indicated by the following example:

```
onReady: function() {
    var myData = [
        {fname: 'Steve', lname:'Drucker'},
        {fname: 'David', lname:'Marsland'}
    ];

    var pnl = new Ext.Panel({
        fullscreen: true,
        data: myData,
        tpl: [
            '<tpl for=".">' ,
            '<div class="fullname">{lname}, {fname}</div>',
            '</tpl>'
        ]
    })
}
```

### Defining Data Templates

Data templates, as illustrated by the prior example, consist of HTML and variable placeholders wrapped by curly braces. Templates may be defined either in-line as with the prior example, or created explicitly using the `Ext.Template()` or `Ext.XTemplate()` constructor. Of the two classes, you will typically use the later as it supports the generation of output from arrays of objects.

As indicated by the prior example, when referencing arrays you will use the `tpl` tag and the `for` operator to process the data object.

The following examples illustrate the use of this construct:

```
<tpl for=".">...</tpl>      // loop through array at root
<tpl for="foo">...</tpl>  // loop through array at foo node
<tpl for="foo.bar">...</tpl> // loop through at foo.bar node
```

In a template, the pound sign (#) represents the current array index plus one.

If you are working with datasets larger than a single record, you may want to define your template and precompile it prior to referencing it in your Panel as indicated by the following example:

```

onReady: function() {
    var myData = [
        {fname: 'Steve', lname:'Drucker'},
        {fname: 'Dave', lname:'Gallerizzo'}
    ];

    var myTemplate = new Ext.XTemplate(
        '<tpl for=".">' ,
        '<div class="fullname">#{#: {lname}, {fname}}</div>' ,
        '</tpl>' ,
        {compiled: true}
    );

    var pnl = new Ext.Panel({
        fullscreen: true,
        data: myData,
        tpl: myTemplate
    })
}
}

```

Output from this code example is:

```
#1: Drucker, Steve
#2: Marsland, David
```

## Formatting Strings in Templates

Sencha Touch supports the following string format functions, which are members of the `Ext.util.Format` class:

Function	Description
<code>date</code>	Parse a value into a formatted date using the specified format pattern.
<code>ellipsis</code>	Truncate a string and add an ellipsis ('...') to the end if it exceeds the specified length
<code>escape</code>	Escapes the string for single quotes and backslashes
<code>escapeRegex</code>	Escapes the passed string for use in a regular expression
<code>format</code>	Allows you to define a tokenized string and pass an arbitrary number of arguments to replace the tokens. Each token must be unique, and must increment in the format {0}, {1}, etc
<code>htmlDecode</code>	Convert certain characters (&, <, >, and ') from their HTML character equivalents.

Function	Description
htmlEncode	Convert certain characters (&, <, >, and ') to their HTML character equivalents for literal display in web pages.
leftPad	Pads the left side of a string with a specified character. This is especially useful for normalizing number and date strings.
toggle	Utility function that allows you to easily switch a string between two alternating values. The passed value is compared to the current string, and if they are equal, the other value that was passed in is returned. If they are already different, the first value passed in is returned. Note that this method returns the new value but does not change the current string.
trim	Strips leading and trailing white space

You can invoke these functions using the following template syntax:

```
{fieldname:functionName [(<params>)]}
```

Note that you can also declare and deploy your own functions as illustrated by the following example:

```
var myTemplate = new Ext.XTemplate(
    '<tpl for=".">' ,
        '<div class="fullname">' ,
            '#{#}: {lname}, {fname} : {age:this.ageGroup}' ,
        '</div>' ,
    '</tpl>' ,
    {compiled: true}
);

myTemplate.ageGroup = function(value) {
    if (value > 1 && value < 13) {
        return "Child";
    } else if (value >= 13 && value < 19) {
        return "Teen";
    } else {
        return "Adult";
    }
}
```

Alternately, you can include the function definition as part of the new Ext.XTemplate constructor:

```
var myTemplate = new Ext.XTemplate(
    '<tpl for=".">' ,
        '<div class="fullname">' ,
            '#{#: {lname}, {fname} : {age:this.ageGroup}}',
            'in five years - {age + 5}' ,
        '</div>' ,
    '</tpl>' ,
    {
        compiled: true,
        ageGroup : function(value) {
            if (value > 1 && value < 13) {
                return "Child";
            } else if (value >= 13 && value < 19) {
                return "Teen";
            } else {
                return "Adult";
            }
        }
    })
});
```

## Processing Output Conditionally using Data Templates

The tpl element also enables conditional processing with basic comparison operators. Note the following:

- Double quotes must be encoded if used within the conditional
- There is no else operator — if needed, two opposite if statements should be used.

```
onReady: function() {
    var myData = [
        {fname: 'Steve', lname:'Drucker', age: 41},
        {fname: 'Dylan', lname:'Drucker', age: 14 },
        {fname: 'Aidan', lname: 'Drucker', age: 5}
    ];

    var myTemplate = new Ext.XTemplate(
        '<tpl for=".">' ,
            '<div class="fullname">' ,
                '#{#: {lname}, {fname} : ',
                '<tpl if="age >=18">Adult</tpl>' ,
                '<tpl if="age > 1 && age <= 13">Child</tpl>' ,
                '<tpl if="age > 13 && age < 18">Teen</tpl>' ,
            '</div>' ,
        '</tpl>' ,
        {compiled: true}
    );
    var pnl = new Ext.Panel({
        fullscreen: true,
        data: myData,
        tpl: myTemplate
    })
}
```

## Executing Basic Arithmetic Expressions in Templates

Your template expressions can include basic arithmetic operators (add, subtract, multiply, divide) as illustrated below:

```
var myTemplate = new Ext.XTemplate(
    '<tpl for=".">' ,
        '<div class="fullname">' ,
            '#{#:} : {lname}, {fname} : {age}' ,
            'in five years - {age + 5}' ,
        '</div>' ,
    '</tpl>' ,
    {compiled: true}
);
```

## Executing inline code

You can insert custom JavaScript expressions into your templates between { [ . . . ] }. Within the block you can reference the following environment variables:

Variable	Description
values	The values in the current scope
parent	The values of the ancestor template
xindex	The index of the loop you are in (1-based)
xcount	The total length of the array that you are looping through.

The following example illustrates using JavaScript expressions in a template:

```
var myTemplate = new Ext.XTemplate(
    '<tpl for=".">' ,
        '<div class="{{xindex % 2 === 0 ? "even" : "odd"}}>' ,
            '#{#:} : {[values.lname.toUpperCase()]}, {fname} : ' ,
            '{[values.age+=5]}' ,
        '</div>' ,
    '</tpl>' ,
    { compiled: true }
);
```

## Walkthrough 5-1: Working with Templates



In this walkthrough, you will perform the following tasks:

- Define a data template
- Output formatted data in a panel



**Illustration 1: Output the contents of a JavaScript array to a Panel**

## Steps

### Review the Data

1. Open [/walk/walk5-1/index.htm](#) in your editor
2. Review lines 9-30 with your instructor

3. Type the following url into your web browser:

```
http://www.senchatraining.com/ftst/components/crimeservice.cfc?  
method=crimereportjson&callback=cachedata
```

4. Review the output with your instructor.
5. Return to your editor

### Define a template

6. Where indicated by the comment, define an **Ext.XTemplate** named **crimeTemplate** as indicated below:

```
var crimeTemplate = new Ext.XTemplate( );
```

7. Inside the **Ext.XTemplate** constructor, define a template that can be used to output the dataset that you reviewed in step 5 and look like the screen shot on the previous page. Your code should resemble the following:

```
var crimeTemplate = new Ext.XTemplate(  
    '<tpl for=".">' ,  
    '<div class="crimereport">' ,  
    '#{#}: {OFFENSE} : {REPORTDATE} <br />' ,  
    '{ADDRESS} <br />' ,  
    '{DESCRIPTION}<br />' ,  
    '</div>' ,  
    '</tpl>' ,  
    {compiled: true}  
> ;
```

### Output Data into the listPnl

8. Add the following configuration parameters passed to your **listPnl** definition to output the dataset using the template that you defined in the previous step
  - data: crimeData
  - tpl: crimeTemplate
  - scroll: 'vertical'
9. Save the file and test. Note that the crime report date also displays the time.

### Format the Date

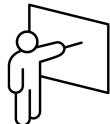
10. Modify your template to output the date in mm/dd/yyyy format. as indicated below:

```
{REPORTDATE:date}
```

11. Save the file and test.

– End of Walkthrough --

# Reading Data from a Server into a Model



Define a Proxy and data Store to read information from a server and cache it in browser memory. Proxies define a connection between the browser and the server. Stores load data via the proxy into data Models and provide functions for sorting, filtering, and querying the model instances contained within it.

There are two types of proxies:

- Client proxies save their data locally (covered in unit 6)
- Server proxies save their data by sending requests to a remote server

There are two types of server proxies:

- AjaxProxy - sends requests to a server on the same domain
- ScriptTagProxy - uses JSON-P to send requests to a server on a different domain

Proxies operate on the principle that all operations performed are either Create, Read, Update or Delete. These four operations are mapped to the methods create, read, update and destroy respectively. Each Proxy subclass implements these functions.

The CRUD methods each expect an operation object as the sole argument. The Operation encapsulates information about the action the Store wishes to perform, the model instances that are to be modified, etc. Each CRUD method also accepts a callback function to be called asynchronously on completion.

Proxies also support batching of Operations via a batch object, invoked by the store.sync() method.

The following code snippet illustrates the use of a simple data storer:

```
// Set up a model to use in our Store
Ext.regModel('User', {
    idProperty: 'userId',
    fields: [
        {name: 'userId', type: 'int'},
        {name: 'firstName', type: 'string'},
        {name: 'lastName', type: 'string'},
        {name: 'age', type: 'int'},
        {name: 'eyeColor', type: 'string'}
    ],
    proxy: {
        type: 'ajax',
        url : '/users.json'
    }
});

var myStore = new Ext.data.Store({
    model: 'User',
    autoLoad: true
});
```

## Defining a Proxy

As previously stated, you can define two types of proxies for making external data requests.

- The `AjaxProxy` is used to send requests to a server on the same domain.
- The `ScriptTagProxy` uses JSON-P to send requests to a server on a different domain.

Both the `AjaxProxy` and the `ScriptTagProxy` extend the `Ext.data.Proxy` class which supports the following configuration options:

Config Option	Description
<code>batchOrder</code>	String. Comma-separated ordering 'create', 'update' and 'destroy' actions when batching. Override this to set a different order for the batched CRUD actions to be executed in. Defaults to 'create,update,destroy'
<code>defaultReaderType</code>	String. The default registered reader type. Defaults to 'json'
<code>defaultWriterType</code>	String. The default registered writer type. Defaults to 'json'

## Defining an AjaxProxy

The `Ext.data.AjaxProxy` class enables you to define a connection between your browser and the domain from which the running page was served.

Key configuration options include the following:

Config Option	Description
<code>headers</code>	Object. Add headers to add to the Ajax request. Defaults to undefined.
<code>noCache</code>	Boolean. Defaults to true. Disable caching by adding a unique
<code>timeout</code>	Number. The number of miliseconds to wait for a response. Defaults to 30 seconds.
<code>url</code>	String. The URL from which to request the data object.

The following example illustrates defining an Ajax proxy that loads data into the associated data model.

```

onReady: function() {
    Ext.regModel('Crime', {
        fields: [
            {name: 'REPORTDATE', type: 'date'},
            {name: 'OFFENSE', type: 'string'},
            {name: 'ADDRESS', type: 'string'}
        ],
        proxy: {
            type: 'ajax',
            url: 'crimeservice.cfc?method=crimesearch'
        },
    });
}

crimeStore = new Ext.data.Store({
    model: 'Crime',
    storeId: 'crimeStore',
    autoLoad: true
});
}
}

```

Note the following:

- The URL is invoking a server-side ColdFusion Component file.
- You could invoke any application server page (ASP.NET, JSP, PHP) so long as it returns data in a JSON format.
- The field names are case-sensitive and must therefore match exactly with the field names output in your JSON data.

## Updating an AjaxProxy Data Set

During the course of execution, you may need to load different information from a server into your Store. In order to accomplish this, you will need to make use of the following methods:

- **Ext.data.Store.setProxy(String|Object|Ext.data.Proxy proxy)**  
Sets the Store's Proxy by string, config object, or Proxy instance
- **Ext.data.Store.load(Object/Function options)**  
Loads data into the Store via the configured proxy. This uses the Proxy to make an asynchronous call to whatever storage backend the Proxy uses, automatically adding the retrieved instances into the Store and calling an optional callback if required.

The following example illustrates changing the URL of a proxy and loading the data into its associated store.

```

crimeStore.setProxy ({
    type: 'ajax',
    url: 'crimeservice.cfc?method=crimesearch&address=' +
        escape("1820 K St NW")
});
crimeStore.load();

```

## Defining a ScriptTag Proxy

`Ext.data.ScriptTagProxy` is an implementation of `Ext.data.Proxy` that reads a data object from a URL which may be in a domain other than the originating domain of the running page.

The content passed back from a server resource requested by a `ScriptTagProxy` must be executable JavaScript source code because it is used as the source inside a `<script>` tag.

In order for the browser to process the returned data, the server must wrap the data object with a call to a callback function, the name of which is passed as a parameter by the `ScriptTagProxy`. Below is a PHP example for which returns data for either a `ScriptTagProxy`, or an `AjaxProxy` depending on whether the callback name was passed:

```
$callback = $_REQUEST['callback'];

// Create the output object.
$output = array('a' => 'Apple', 'b' => 'Banana');

//start output
if ($callback) {
    header('Content-Type: text/javascript');
    echo $callback . '(' . json_encode($output) . ')';
} else {
    header('Content-Type: application/x-json');
    echo json_encode($output);
}
```

Configuration attributes for a `ScriptTagProxy` include the following:

Config Param	Description
callbackParam	<p>String. The name of the parameter to pass to the server which tells the server the name of the callback function set up by the load call to process the returned data object. Defaults to "callback".</p> <p>The server-side processing must read this parameter value, and generate javascript output which calls this named function passing the data object as its only parameter.</p>
callbackPrefix	The prefix string that is used to create a unique callback function name in the global scope. This can optionally be modified to give control over how the callback string passed to the remote server is generated. Defaults to 'stcCallback'
url	String. The URL from which to request the data object.

The following code snippet illustrates making a scripttag proxy call:

```

onReady: function() {
    Ext.regModel('Crime', {
        fields: [
            {name: 'REPORTDATE', type: 'date'},
            {name: 'OFFENSE', type: 'string'},
            {name: 'ADDRESS', type: 'string'}
        ],
        proxy: {
            type: 'scripttag',
            url: 'http://127.0.0.1:8100/crimeservice.cfc?',
            method='crimesearchJSONP'
        }
    });
}

crimeStore = new Ext.data.Store({
    model: 'Crime',
    storeId: 'crimeStore',
    autoLoad: true
});
}

```

The corresponding server-side code in ColdFusion is the following:

```

<cffunction name="crimesearchJSONP"
    access="remote"
    returntype="any"
    returnformat="plain"
    output="false"
    hint="returns result in json-p format">

    <cfargument name="callback" type="string">
    <cfargument name="address"
        type="string" default="1400 16th Street NW">
    <cfargument name="radius" type="numeric" default="0.1">

    <cfset var aData =
        crimeDataGet(arguments.address,arguments.radius)>

    <cfset adata = arguments.callback & "(" &
        serializeJSON(aData) & ")">

    <cfcontent type="text/javascript">
    <cfreturn aData>
</cffunction>

```

## Defining a Store

Define a Store using the Ext.data.Store constructor as indicated by the previous example. Store configuration options include the following:

Config Option	Description
autoLoad	Boolean. If data is not specified, and if autoLoad is true or an Object, this store's load method is automatically called after creation. If the value of autoLoad is an Object, this Object will be passed to the store's load method. Defaults to false.
data	Array. Optional array of Model instances or data objects to load locally.
proxy	The Proxy to use for this Store. This can be either a string, a config object or a Proxy instance. It overrides the proxy definition set within the model.
sorters	Array. Defines the client-side sorting fields and direction.
filters	Array. Defines filter criteria.
storeId	Optional unique identifier for this store. If present, this Store will be registered with the Ext.StoreMgr, making it easy to reuse elsewhere.
remoteFilter	Boolean. True to defer any filtering operation to the server. If false, filtering is done locally on the client. Defaults to false. Set to true if using large data sets.
remoteSort	True to defer sorting to the server. If false, sorting is done locally on the client. Defaults to false. Set to true if using large data sets.

## Using Inline Data

Stores can load data inline. Internally, Store converts each of the objects into Model instances:

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

crimeStore = new Ext.data.Store({
    model: 'Crime',
    storeId: 'crimeStore',
    data : [
        {id: 1, offense: 'Theft', address: '1400 16th St NW'},
        {id: 2, offense: 'Assault', address: '1523 16th St NW'},
        {id: 3, offense: 'Drug Pos', address: '1201 15th St NW'},
        {id: 4, offense: 'Auto Theft', address: '1801 23rd St NW'},
    ]});

```

## Sorting data in your store

You can sort the data in your store using the `sort()` method as indicated below:

```
crimeStore.sort('id', 'ASC');
```

If you need to sort by multiple fields, simply pass an array of sorters to the `sort()` method as described below:

```
crimeStore.sort([
    {property: 'offense', direction: 'ASC'},
    {property: 'address', direction: 'DESC'}
]);
```

Note that all existing sorters will be removed in favor of the new sorter data (if `sort()` is called with no arguments, the existing sorters are just reapplied instead of being removed). To keep existing filters and add new ones, use the `sorters.add()` method as indicated by the following example:

```
crimeStore.sorters.add(new Ext.util.Sorter({
    property : 'id',
    direction: 'ASC'
}));

store.sort();
```

## Filtering data in the Store

Conversely, you can filter data using the `filter()` method described below. Note that invoking `filter()` is a non-destructive operation.

```
crimeStore.filter('offense', 'Theft');
```

Calling `filter()` with no arguments re-applies any existing filters. Use the `clearFilter()` method to revert to a view of the record cache with no filtering applied.

## Understanding Server Data Requests

When you make a `store.load()` request, or execute a `store.sort()` or `store.filter()` method with the `remoteSort` and `remoteFilter` store properties set to true, Sencha Touch automatically adds the following URL arguments to your proxy URL:

URL Variable	Description
<code>_dc</code>	A random number, used to ensure that the browser does not read the data from its cache.
<code>callback</code>	If using a scripttag proxy, this contains the name of the callback function that you must use as a wrapper for your JSON data set.
<code>sort</code>	If any sorters are defined, or you have set <code>remoteSort</code> equal to true in your store, this variable contains your sort definition in URL encoded JSON format. For example:  <code>sort=[{"property": "OFFENSE", "direction": "ASC"}]</code>
<code>filter</code>	If you set <code>remoteFilter</code> to true in your store definition, your filter definition will be sent to the server as URL encoded JSON string. For example:  <code>filter=[{"property": "offense", "value": "Theft"}]</code>
<code>limit</code>	Used for data pagination. Contains the total number of records that are being requested from the server. Defaults to 25. You can change this value by setting <code>Store.pageSize</code> .
<code>start</code>	Using the <code>Store.loadPage(pagenum)</code> method causes this URL parameter to be transmitted, representing the first data record to be returned from the server. Use in conjunction with the <code>limit</code> parameter in your SQL.

## Grouping Data in the Store

Models that contain a common field value may be grouped together by specifying the following two store configuration attributes:

Config	Description
groupField	The field by which to group data in the store. Internally, grouping is very similar to sorting - the <code>groupField</code> and <code>groupDir</code> are injected as the first sorter. Stores support a single level of grouping, and groups can be fetched via the <code>getGroups()</code> method.
groupDir	The direction in which sorting should be applied when grouping. Defaults to "ASC" - the other supported value is "DESC"

The following example illustrates configuring a store for grouping as well as generating an array containing the count of the members for each group. Note that the `Ext.data.Store.getGroups()` method returns an Array of Objects with the following keys:

- `name` : Contains the value for each group
- `children` : An array of model instances that belong to the group

```

var crimeStore = new Ext.data.Store ({
    storeId: 'crimeStore',
    model: 'CrimeReport',
    autoLoad: false,
    remoteFilter: true,
    remoteSort: true,
    groupField: 'OFFENSE',
    groupDir: 'ASC'
});

var crimeStats = [];

crimeStore.load({
    callback: function(records, op, success) {
        var aGroups = crimeStore.getGroups();
        for (var i = 0; i < aGroups.length; i++) {
            crimeStats.push({
                label: aGroups[i].name,
                count: aGroups[i].children.length
            });
        } /* for */
    } /* callback */
}); /* load() */

```

## Outputting the Contents of a Model

Data stored within models have a very specific data structure and attached event handlers. While you cannot output the content stored within a model using the techniques previously discussed, there are alternatives – notably the DataView (Ext.DataView) and its associated subclasses (Ext.List).

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

*Note: Set the containing panel's layout attribute to 'fit' in order to contain the entire contents of a DataView.*

The following example illustrates using a DataView to output the contents of a Model:

```
var store = new Ext.data.JsonStore({
    url: 'get-images.php',
    root: 'images',
    fields: [
        {name: 'name', type: 'string'},
        {name: 'url', type: 'string'},
        {name: 'size', type: 'float'},
        {name: 'lastmod', type: 'date', dateFormat:'timestamp'}
    ]
});
store.load();

var tpl = new Ext.XTemplate(
    '<tpl for=".">' ,
    '<div class="thumb-wrap" id="{name}">',
    '<div class="thumb"></div>',
    '<span class="x-editable">{shortName}</span></div>',
    '</tpl>',
    '<div class="x-clear"></div>'
);
var panel = new Ext.Panel({
    layout:'fit',
    title:'Simple DataView',
    items: new Ext.DataView({
        store: store,
        tpl: tpl,
        autoHeight:true,
        multiSelect: true,
        overClass:'x-view-over',
        itemSelector:'div.thumb-wrap',
        emptyText: 'No images to display'
    })
});
```

## Using DataView Configuration Properties

Ext.DataView extends Ext.Component and has the following class-specific configuration properties:

Config Option	Description
allowDeselect	Boolean. Only respected if singleSelect is true. If this is set to false, a selected item will not be deselected when tapped on, ensuring that once an item has been selected at least one item will always be selected. Defaults to allowed (true).
blockRefresh	Boolean. Set this to true to ignore datachanged events on the bound store. This is useful if you wish to provide custom transition animations via a plugin (defaults to false)
deferEmptyText	Boolean. True to defer emptyText being applied until the store's first load.
disableSelection	Boolean. True to disable selection within the DataView. Defaults to false. This configuration will lock the selection model that the DataView uses.
emptyText	String. The text to display in the view when there is no data to display (defaults to "").
itemSelector	String. The css class that wraps an element in an Ext.XTemplate. (example: "div.myrecord")
loadingText	String. Text to display during data load operations (defaults to undefined). If specified, this text will be displayed in a loading div and the view's contents will be cleared while loading, otherwise the view's contents will continue to display normally until the new data is loaded and the contents are replaced.
multiSelect	Boolean. True to allow selection of more than one item at a time. False to allow selection of only one item at a time.
overItemCls	A CSS class to apply to each item in the view on mouseover
pressedCls	A CSS class to apply to an item on the view while it is being pressed.

Config Option	Description
pressedDelay	Number. The amount of delay between the tapstart and when pressedCls is applied. Default is 100ms.
selectedItemCls	String. A CSS class to apply to each selected item in the view (defaults to 'x-view-selected').
simpleSelect	Boolean. True to enable multiselection by clicking on multiple items without requiring the user to hold Shift or Ctrl, false to force the user to hold Ctrl or Shift to select more than one item (defaults to false).
singleSelect	Boolean. True to allow selection of exactly one item at a time, false to allow no selection at all (defaults to false). Note that if multiSelect = true, this value will be ignored.
store	The Ext.data.Store instance that is used to populate the output.
tpl	The Ext.Template used to format the data contained in the store.
trackOver	Boolean. True to enable mouseenter and mouseleave events.
triggerEvent	Defaults to 'singletap'. Other valid options are 'tap'

## Walkthrough 5-2: Working with Models, Stores, and Proxies



In this walkthrough, you will perform the following tasks:

- Define a proxy to retrieve information from the server via AJAX
- Load a dataset from a server into a Model
- Inspect data using debugging tools in Safari
- Sort data in a model
- Output model data into a DataView



### Steps

#### Review the Starting Code and Data

1. Using your editor, open [/walk/walk5-2/index.htm](#). Note that this is essentially the solution from the prior walkthrough
2. Delete lines **9-17**
3. Open a web browser to the following url:

```
http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=crimesearchjsonp&callback=foo
```

4. Review the structure of the data with your instructor

#### Illustration 2: Implementing a DataView

### Define a Data Model

5. Where indicated by the comment, define a model named **CrimeReport** with the following fields:

Field	Type
REPORTDATE	date
OFFENSE	string
ADDRESS	string
DESCRIPTION	string

6. Add a proxy configuration attribute to your model that reads data from the following URL:

```
http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=crimesearchjsonp
```

7. Review your code to ensure it is similar to the following:

```
Ext.regModel('CrimeReport', {
    fields: [
        {name: 'REPORTDATE', type: 'date'},
        {name: 'OFFENSE', type: 'string'},
        {name: 'ADDRESS', type: 'string'},
        {name: 'DESCRIPTION', type: 'string'}
    ],
    proxy: {
        type: 'scripttag',
        url : '../json/crimesearch.json',
        reader: {
            type: 'json'
        }
    }
});
```

8. Where indicated by the comment, define a public variable named **CrimeStore** that points to a new instance of a data Store. Your code should appear similar to the following:

```
crimeStore = new Ext.data.Store({});
```

9. Modify the configuration properties of the data Store to use the model that you defined in steps 5-7 and configure it to automatically load data on startup. Your code should appear similar to the following:

```
crimeStore = new Ext.data.Store({
    model: 'CrimeReport',
    storeId: 'crimeStore',
    autoLoad: true
});
```

10. Immediately after the crimeStore instantiation, sort the contents of the Store by OFFENSE as indicated below:

```
crimeStore.sort('OFFENSE', 'ASC');
```

#### Inspect the data in Safari.

11. Browse the file in Safari
12. From the Safari menu, select **Develop > Show Error Console**
13. Click on the **Scripts** button in the Error Console
14. In the **Watch Expressions** panel, click the **Add** button
15. Enter **crimeStore** as the variable name
16. Click the **Refresh** button
17. Click through the **CrimeStore** variable structure to reach **crimeStore > data > items**. You should see 74 items listed. Review the data with your instructor

#### Sort the Data

18. Modify the segmented button handlers in the listPnl panel to sort the data in the store by **OFFENSE** in **ascending** order and **REPORTDATE** in **descending** order. Your code should be similar to the following:

```
xtype: 'segmentedbutton',
items: [
{
    text: 'Type',
    iconCls: 'organize',
    iconMask: true,
    handler: function()
        {crimeStore.sort('OFFENSE', 'ASC')}
},
{
    text: 'Date',
    iconCls: 'organize',
    iconMask: true,
    handler: function()
        {crimeStore.sort('REPORTDATE', 'DESC')}
}
]
```

19. Save the file and test in Safari. Return to the **Error Console** and inspect the data in the Store.
20. In your application, click on one of the sort by date button that you modified in step 16.
21. Refresh the data in the **Watch Expression** panel. Note how the order of the records has changed.

### Output the Data into a DataView

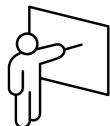
22. Add the following configuration properties to the listPnl declaration:
  - layout: 'fit'
23. In the listPnl declaration, add an items attribute to the configuration properties that outputs the contents of your store using an Ext.DataView() constructor. Your code should resemble the following:

```
items: new Ext.DataView({});
```

24. Set the following configuration properties for Ext.DataView:
  - **store:** crimeStore
  - **tpl:** crimeTemplate
  - **singleSelect:** true
  - **scroll:** 'vertical'
  - **itemSelector:** 'div.crimereport'
  - **emptyText:** 'No crimes were found'
  - **selectedItemCls:** 'crimereportSelected'
25. Save the file and test. Note that pressing either of the sort buttons causes the DataView to automatically update.

– End of Walkthrough --

# Displaying Data in a List



The list class (Ext.List) extends DataView. It uses an Ext.XTemplate as its internal templating mechanism, and is bound to an Ext.data.Store so that as the data in the store changes the view is automatically updated to reflect the changes.

Use a List instead of a DataView for instances where you have lists of names, data that may be grouped, or if you want to render an alphabet IndexBar docked on the right.

The following example demonstrates a simple instantiation of an Ext.List:

C	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Aaron Conran	
E	
Abraham Elias	
K	
Dave Kaneda	
M	
Tommy Maintz	
P	
Michael Mullany	

Illustration 3: An implementation of Ext.List

```

Ext.regModel('Contact', {
    fields: ['firstName', 'lastName']
});

var store = new Ext.data.JsonStore({
    model : 'Contact',
    sorters: 'lastName',

    getGroupString : function(record) {
        return record.get('lastName')[0];
    },

    data: [
        {firstName: 'Aaron', lastName: 'Conran'},
        {firstName: 'Dave', lastName: 'Kaneda'},
        {firstName: 'Michael', lastName: 'Mullany'},
        {firstName: 'Abraham', lastName: 'Elias'},
        {firstName: 'Jay', lastName: 'Robinson'},
        {firstName: 'Tommy', lastName: 'Maintz'},
    ]
});

var list = new Ext.List({
    tpl: '<tpl for=".">><div class="contact">{firstName}<br/><strong>{lastName}</strong></div></tpl>',

    itemSelector: 'div.contact',
    singleSelect: true, grouped: true,
    indexBar     : true,

    store: store,

    floating      : true, width : 350, height : 370,
    centered       : true, modal : true,
    hideOnMaskTap: false
});
list.show();

```

## Configuring an Ext.List

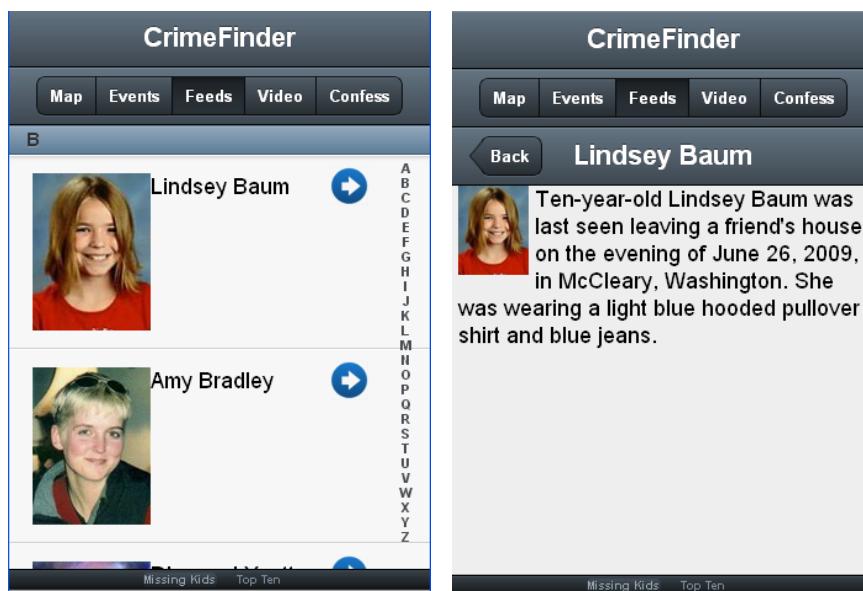
The following configuration properties are specific to Ext.List:

Config Option	Description
clearSelectionOnDeactivate	Boolean. True to clear any selections on the list when the list is deactivated (defaults to true).
grouped	Boolean. True to group the list items together (defaults to false). When using grouping, you must specify a method <code>getGroupString</code> on the store so that grouping can be maintained.
indexBar	Boolean/Object. True to render an alphabet IndexBar docked on the right. This can also be a config object that will be passed to Ext.IndexBar (defaults to false)
itemTpl	The inner portion of the item template to be rendered. Follows an XTemplate structure and will be placed inside of a <code>tpl</code> for in the <code>tpl</code> configuration.
onItemDisclosure	Boolean/Function/Object. True to display a disclosure icon on each list item. This won't bind a listener to the tap event. The list will still fire the disclose event though. By setting this config to a function, it will automatically add a tap event listeners to the disclosure buttons which will fire your function. Finally you can specify an object with a 'scope' and 'handler' property defined. This will also be bound to the tap event listener and is useful when you want to change the scope of the handler.
pinHeaders	Whether or not to pin headers on top of item groups while scrolling for an iPhone native list experience Defaults to true

## Creating an ItemDisclosure Handler

Typically you will want to create applications where clicking on a list item provides the user with additional detailed information about the record. In Sencha Touch terminology, this is referred to as an “item disclosure.” It is also commonly referred to as “data drill-down.”

One approach to implementing this functionality is to specify an itemDisclosure function for your list which will add an icon to the right of each list element and invoke your handler when the icon is touched as indicated by the screen shots below:



In the prior screen shots, the application is organized into a parent panel containing two nested panels – the list displaying the children's names and photos as well as the detailed information which is populated at runtime. The items organizational structure therefore resembles the following:

```
var containerPanel = new Ext.Panel({
    items: [
        new Ext.List({ ... }), // children photos
        {
            xtype: 'panel',
            id: 'missingChildDetail',
            title: 'Missing Children',
            html: 'Placeholder'
        }
    ]
})
```

Once the panel layout has been implemented you will need to define an itemDisclosure handler function accepts that accepts the following parameters, listed in order:

- record : Ext.data.Model  
The model instance associated with the item
- node : Ext.Element  
The wrapping element of this node
- index : Number  
The index of this list item

Your itemDisclosure handler would subsequently use the following methods to populate and display the 'missingChildDetail' panel:

- Ext.getCmp(string ID)  
Returns the Sencha Touch object associated with the specified ID
- Ext.data.Model.get(string fieldname)  
Returns the data associated with the name of the field for a model instance
- Ext.Panel.update(string html)  
Updates the contents of a panel with the specified HTML
- Ext.Toolbar.setTitle(string)  
Updates the title of the toolbar
- Ext.Panel.setActiveItem (int index)  
Brings the specified panel into view

Therefore, a typical itemDisclosure handler might resemble the following:

```
onItemDisclosure: function (objRec, objNode, recIndex) {  
    /* populate the detail page */  
    Ext.getCmp('missingChildDetail').update('<div  
class="kidDetail">' + objRec.get('DESCRIPTION') + '</div>');  
  
    /* change the detail page toolbar title */  
    Ext.getCmp('kidsDetailToolbar').setTitle(objRec.get('TITLE'));  
    /* bring the detail panel forward */  
    Ext.getCmp('kidsPanel').setActiveItem(1);  
}
```

## Walkthrough 5-3: Using a List for Data Drilldown



In this walkthrough, you will perform the following tasks:

- Define a proxy to retrieve information from the server via AJAX
- Load a dataset from a server into a Model
- Display data from a Model in a List
- Create a data drill-down page

The image consists of two side-by-side screenshots of a mobile application. Both screenshots have a header bar with tabs: 'Map', 'Events', 'Feeds', 'Video', and 'Confess'. Below the header is a search bar containing the letter 'C'. The left screenshot displays a list of missing children. The first item in the list is 'Amber Elizabeth Cates', accompanied by a photo of a young girl. To the right of the list is a vertical scroll bar with letters A through Z. The right screenshot shows a detailed view for 'Allyson Corrales', featuring a photo of a young girl and a descriptive text block: 'Allyson Corrales has been missing from her residence in Kansas City, Missouri, since March 6, 2009. She may be in the company of her father, Luis Corrales. Allyson's mother, who was found deceased on March 6, 2009, was not married to Luis Corrales. He did not have any custodian rights to Allyson and the mother had a Full Order of Protection against him.' At the bottom of both screenshots are navigation buttons labeled 'Back', 'Missing Kids', and 'Top Ten'.

**Illustration 4: The Missing Children RSS Feed from the FBI**

## Steps

### Review the Starting Code and Data

1. Using your editor, open `/walk/walk5-3/index.htm`. Note that this is essentially the solution from the prior walkthrough with a few new CSS classes added.
2. Open a web browser to the following url:  
`http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=getmissingchildrenjsonp&callback=foo`
3. Review the structure of the output

### Define the Data Model

4. Where indicated by the comment, define a data model that pulls data from the URL and matches with the structure of the JSON output that you reviewed in step 2. Name the model **MissingKids**
5. Review your data model to ensure that it resembles the following:

```
Ext.regModel('MissingKids', {
    fields: [
        {name: 'TITLE', type: 'string'},
        {name: 'FIRSTNAME', type: 'string'},
        {name: 'LASTNAME', type: 'string'},
        {name: 'DESCRIPTION', type: 'string'},
        {name: 'PHOTO', type: 'string'}
    ],
    proxy: {
        type: 'scripttag',
        url:
            'http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=getmissingchildrenjsonp'
    }
});
```

### Define a Data Store to Fetch the Data

6. Define a variable named **missingKidsStore** that points to a new **Ext.data.Store()**
  - The Store should reference the Model that you defined in step 4
  - Use **missingKidsStore** as the Store ID
  - Sort the data in the store by last name
  - The store should automatically load the data
7. Verify that your code appears similar to the following:

```
missingKidsStore = new Ext.data.Store({
    model: 'MissingKids',
    storeId: 'missingKidsStore',
    sorters: [{property : 'LASTNAME', direction: 'ASC'}],
    autoLoad: true
});
```

8. Add another attribute to your Store named **getGroupString** that returns the first character of the record's **LASTNAME** field as indicated below:

```
getGroupString : function(record) {
    return record.get('LASTNAME')[0];
}
```

### Output the contents of the Model to a List

9. Under the comment for step 9, define an **items** configuration attribute for the panel
10. Define an **Ext.List** as the first item in the array that you declared in the prior step. use the following attributes:
  - **store:** missingKidsStore
  - **itemTpl:** '<div class="contact">{TITLE}</div>'
  - **grouped:** true
  - **indexBar:** true
  - **pinHeaders:** true
  - **clearSelectionOnDeactivate:** true
11. Save the file and browse. You should be able to view the FBI's list of missing children.

### Add the Details Page

12. Append another panel to the items array that you defined in step 8. Use the following attributes:
  - **xtype:** 'panel'
  - **id:** 'missingChildDetail'
  - **title:** 'Missing Children'
  - **html:** 'placeholder'
13. Add a docked toolbar to the panel that you defined in a previous step that contains a title and a back button (refer to the screen shot on page 27 for guidance). Give it an id of 'kidsDetailToolbar'Your code should resemble the following:

```
dockedItems: [
  {
    id: 'kidsDetailToolbar',
    xtype: 'toolbar',
    title: '[name]',
    dock: 'top',
    items: [
      {
        text: 'Back', ui: 'back',
      }
    ]
  }
]
```

### Add the Navigation Handlers

14. Add an **onItemDisclosure** attribute to your **Ext.List** configuration object. Have it invoke a function that performs the following tasks:
  - Update the content of the **missingChildDetail** panel to be the **DESCRIPTION** field of the selected record
  - Change the **title** of the toolbar in the **missingChildDetail** panel to the **TITLE** field of the selected record
  - Make the **missingChildDetail** panel active

15. Validate that your code appears similar to the following:

```
onItemDisclosure: function (objRec, objNode, recIndex) {  
  
    Ext.getCmp('missingChildDetail').update(  
        '<div class="kidDetail">' + objRec.get('DESCRIPTION') +  
        '</div>');  
  
    Ext.getCmp('kidsDetailToolbar').setTitle(  
        objRec.get('TITLE'));  
  
    Ext.getCmp('kidsPanel').setActiveItem(1);  
}
```

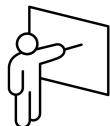
16. Save the file and test. You should be able to click on a record and see detailed information about a case.
17. Return to your editor.
18. Add a handler function for the **Back** button on the details page to return the user back to the list of missing children. Your code should appear as follows:

```
handler: function() {  
    Ext.getCmp('kidsPanel').setActiveItem(0, {  
        type: 'slide',  
        direction: 'right'  
    });  
}
```

19. Save the file and test

– End of Walkthrough –

## Displaying Hierarchical Data in a Nested List



The nested list class, `Ext.NestedList`, extends Panel. It uses an `Ext.data.TreeStore` in order to automatically create drill-down navigation as depicted by the following screen shots and associated code sample.

Groceries	Drinks	Water
Drinks	Water	Sparkling
	Water	Still
	Coffee	

Illustration 5: The progression of clicking through a nested list

```

onReady: function() {

    var data = {
        text: 'Groceries',
        items: [
            {
                text: 'Drinks',
                items: [
                    {
                        text: 'Water',
                        items: [{text: 'Sparkling',leaf: true}, {text: 'Still', leaf: true }]
                    },
                    {text: 'Coffee', leaf: true}
                ]
            }
        ];
    };

    Ext.regModel('ListItem', {
        fields: [{name: 'text', type: 'string'}]
    });

    var store = new Ext.data.TreeStore({
        model: 'ListItem',
        root: data,
        proxy: {
            type: 'ajax',
            reader: {
                type: 'tree',
                root: 'items'
            }
        }
    });
    var nestedList = new Ext.NestedList({
        fullscreen: true,
        title: 'Groceries',
        displayField: 'text',
        store: store
    });
}

```

## Using the TreeStore

The `Ext.data.TreeStore` class allows the representation of hierarchical data. It also enables you to incrementally fetch data from a proxy in order to reduce the latency associated with front-loading a large data set.

`Ext.data.TreeStore` extends `Ext.data.AbstractStore` and supports the following class-specific configuration properties:

Config Option	Description
<code>clearOnLoad</code>	Boolean. Defaults to <code>true</code> . Remove previously existing child nodes before loading.
<code>defaultRootId</code>	String. The default root id passed to the proxy. Defaults to ' <code>root</code> '
<code>nodeParam</code>	String. The name of the parameter sent to the server that contains the identifier of the node. Defaults to ' <code>node</code> '

The following example illustrates defining a tree store and related proxy:

```
Ext.regModel('myModel', {
    fields: [{name: 'text', type: 'string'}]
});
myStore = new Ext.data.TreeStore({
    model: 'myModel',
    storeId: 'mystore',
    proxy: {
        type: 'ajax',
        url : '../../../../../components/get10.cfc?method=getTopTen',
        reader: {
            type: 'tree',
            root: 'items'
        }
    },
    autoLoad: true
});
```

Note the following details regarding the example:

- Sencha Touch automatically passes a URL variable named “node” with an associated value of “root” to the `get10.cfc` file
- Data requested from the URL must be in JSON
- The reader type of “tree” refers to the `Ext.data.Tree` class
- The URL must return a nested dataset, where an `items[]` array denotes child nodes and a `leaf` attribute denotes a terminal node as illustrated by the following:

```
{ text: 'Groceries', items:[{text: 'Milk', leaf: true},
                           {text: 'Eggs', leaf: true}]
}
```

## Making Ad-Hoc AJAX Requests

Typically, when working with a nested list you will want to retrieve detailed information about a record when a user taps on a leaf node. You can make ad-hoc requests to your application server using the Ext.Ajax.request() method which supports the following configuration properties:

Config Option	Description
url	String. The url of the request
success	Function. A callback that is invoked when the request has successfully completed. It is passed a single argument representing the HTTP response object.
failure	Function. A callback that is invoked if the request fails.

Before you invoke the Ext.Ajax.request() method, you should invoke the  `setLoading(true)` method of your visible panel to inform the user of a slight delay while the transaction takes place. In your success and failure handlers you invoke the  `setLoading(false)` method to turn off the wait message.

Note that you can invoke the Ext.decode() method in order to parse complex data that is represented as JSON.

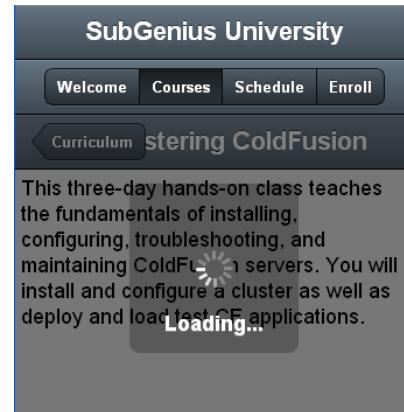


Illustration 6: The result of calling Ext.Panel.setLoading(true)

The following example illustrates making an AJAX request:

```
myPanel.setLoading(true);

Ext.Ajax.request({
    url: 'getdata.php',
    success: function(response) {
        myPanel.update(response.responseText);
        myPanel.setLoading(false);
    },
    failure: function() {
        myPanel.update("Loading failed.");
        myPanel.setLoading(false);
    }
});
```

## Making Ad-Hoc JSON-P Requests

You can also retrieve detailed information about a record through a cross-domain request by using the Ext.util.JSONP.request() method which supports the following configuration properties:

Config Option	Description
url	String. The url of the request
params	Object. A set of key/value pairs to be url encoded and passed as GET parameters in the request
callbackKey	String. Key specified by the server-side provider
callback	Function. Will be passed a single argument of the result of the request
scope	Scope. Scope to execute your callback within.

Before you invoke the Ext.util.JSONP.request() method, you should invoke the panel.  `setLoading(true)` method of your visible panel to inform the user of a slight delay while the transaction takes place. In your callback function you invoke the  `setLoading(false)` method to turn off the wait message.

Unlike its AJAX counterpart, this method does not require that you invoke the `Ext.decode()` method in order to parse the returned data.

The following example illustrates making an Ext.util.JSONP.request:

```
myPanel.setLoading(true);

Ext.util.JSONP.request({
    url: 'http://senchatraining.com/getdata.cfc',
    callbackKey: 'callback', // name of callback url param
    params: { // transmit URL parameters
        method: 'getclassinfojsonp',
        courseid: r.get('courseid')
    },
    callback: function(response) {
        Ext.Msg.alert(response.message);
        myPanel.setLoading(false);
    }
});
```

## Defining a Nested List

Ext.NestedList extends Ext.Panel and had the following class-specific attributes:

Config Option	Description
allowDeselect	Boolean. Set to true to allow the user to deselect leaf items via interaction. Defaults to false.
backText	The label to display for the back button. Defaults to "Back"
cardSwitchAnimation	String. The animation to be used during transitions of cards.
clearSelectionDelay	Boolean/Number. Number of milliseconds to show the highlight when going back in a list. (Defaults to 200). Passing false will keep the prior list selection.
displayField	String. Display field to use when setting item text and title. This configuration is ignored when overriding getItemTextTpl or getTitleTextTpl for the item text or title. (Defaults to 'text')
emptyText	String. Empty text to display when a subtree is empty.
getTitleTextTpl	Function. Return the text to be used as the title
getItemTextTpl	Function. Return the data to display in the panel.
getDetailCard	Function. Accepts two parameters – the selected record and its parent record. Should return a Sencha Touch panel element to display detailed information about the selected leaf node.
loadingText	String. Loading text to display when a subtree is loading.
onItemDisclosure	Maps to the Ext.List onItemDisclosure configuration for individual lists. (Defaults to false)
store	The Ext.data.TreeStore to bind to the NestedList

Config Option	Description
toolbar	Configuration for an Ext.Toolbar created within the Ext.NestedList.
updateTitleText	Update the title with the currently selected category. Defaults to true.
useTitleAsBackTest	Boolean. Set to true use the title as the back button. Defaults to false

The following example illustrates defining a nested list that makes an AJAX request if a leaf node is pressed by the user:

```

var store = new Ext.data.TreeStore({
    model: 'File',
    proxy: {
        type: 'ajax',
        url: 'getSourceFiles.php',
        reader: {
            type: 'tree',
            root: 'children'
        }
    }
});

var nestedList = new Ext.NestedList({
    fullscreen: true,
    title: 'src/',
    displayField: 'fileName',

    getTitleTextTpl: function() {
        return '{' + this.displayField + '}<tpl if="leaf !== true">/</tpl>';
    },

    getItemTextTpl: function() {
        return '{' + this.displayField + '}<tpl if="leaf !== true">/</tpl>';
    },

    getDetailCard: function(record, parentRecord) {
        nestedList.setLoading(true);
        return new Ext.Panel({scroll: 'vertical', layout: {type: 'fit'}})
    },
    store: store
});

nestedList.on('leafitemtap', function(subList, subIdx, el, e, detailCard) {
    var ds = subList.getStore(),
        r = ds.getAt(subIdx);
    Ext.Ajax.request({
        url: '../../src/' + r.get('id'),
        success: function(response) {
            detailCard.update(response.responseText);
            nestedList.setLoading(false)
        },
        failure: function() {
            detailCard.update("Loading failed.");
            nestedList.setLoading(false);
        }
    });
});

```

## Walkthrough 5-4: Using a Nested List for Data Drilldown



In this walkthrough, you will perform the following tasks:

- Define a proxy to retrieve information from the server via AJAX
- Load a hierarchical dataset from a server into a Model
- Display data from a Model in a nested list

CrimeFinder	
<a href="#">Map</a>	<a href="#">Events</a>
<a href="#">Feeds</a>	<a href="#">Video</a>
<a href="#">Confess</a>	
<b>Top 10</b>	
Fugitives	
Terrorists	
<a href="#">Missing Kids</a> <a href="#">Top 10</a>	

CrimeFinder	
<a href="#">Map</a>	<a href="#">Events</a>
<a href="#">Feeds</a>	<a href="#">Video</a>
<a href="#">Confess</a>	
<a href="#">Top 10</a>	<b>Fugitives</b>
	Semion Mogilevich
	Eduardo Ravelo
<a href="#">Missing Kids</a> <a href="#">Top 10</a>	

## Steps

### Review the Starting Code and Data

1. Using your editor, open [/walk/walk5-4/index.htm](#). Note that this is essentially the solution from the prior walkthrough with a new CSS selector that has been added.
2. Open a web browser to the following url:  
<http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=getTopTenJsonP&callback=foo>
3. Review the structure of the output

**Define a Data Model to Contain the Web Service Data**

4. Where indicated by the comment, define a new Model named **top10Model**
5. Associate a single field named **text** of type **string** with your Model
6. Verify that your code appears similar to the following:

```
Ext.regModel('top10Model', {  
    fields: [{name: 'text', type: 'string'}]  
});
```

7. Define a proxy for the store that you defined in the prior step. Use the following attributes:
  - **type:** 'scripttag'
  - **url:**  
`'http://www.senchatraining.com/ftst/components/crimeservice.cfc?method=getoptenjsonp'`
  - **reader:** {type: 'tree', root: 'items'}
8. Where indicated by the comment, define a TreeStore with the following attributes:
  - **model:** 'top10Model'
  - **storeId:** 'top10store'
  - **autoLoad:** true
9. Verify that your TreeStore definition matches the following:

```
top10store = new Ext.data.TreeStore({  
    model: 'top10Model',  
    storeId: 'top10store',  
    autoLoad: true  
});
```

10. Replace the placeholder panel definition located underneath the comment for step 10 with a NestedList that uses the following configuration attributes:
  - **id:** 'top10'
  - **title:** 'Top 10'
  - **iconCls:** 'team'
  - **displayField:** 'text'
  - **store:** top10store
11. Save the file and test

– End of Walkthrough –

## Unit Summary



- Use data templates to output structured data
- You can bind functions to data templates in order to format strings, conditionally output data, execute basic arithmetic expressions, and execute inline JavaScript
- Define a model with a proxy to load data from an application server using AJAX.
- Use an AJAX proxy to fetch information from a server on the same domain as where your application is hosted.
- Use a ScriptTag proxy to fetch information from a server on a different domain than where your application is hosted.
- Use an Ext.List() element to quickly output an array
- Use Ext.NestedList() to output information that is structured in a hierarchical tree

## Unit Review



1. Describe scenarios when outputting data to an Ext.List is more appropriate than using an Ext.NestedList or an Ext.DataPanel?
2. What are the different types of proxies that are available and when would you use each?
3. You can only read JSON from a proxy, not XML (true/false)
4. How can you programmatically change the HTML output in a Panel?
5. List the benefits of loading data into a Store.

## Lab 5: Displaying a List of Courses



During this lab you will develop the code necessary to display a contact list of instructors as well as output information about the various courses offered by SubGenius University.

SubGenius University	
<a href="#">Welcome</a>	<a href="#">Courses</a>
<a href="#">Schedule</a>	<a href="#">Enroll</a>
<a href="#">By Name</a>	<b>Professors</b>
<a href="#">By Specialty</a>	
<b>B</b> Backer,Lisa Adobe Flex lbacker@figleaf.com 555-555-CODE	
<b>D</b> Drucker,Steve Sencha Touch sdrucker@figleaf.com 202-415-8483	
A B C D E F G H I J K L M N O P Q R S T	

SubGenius University	
<a href="#">Welcome</a>	<a href="#">Courses</a>
<a href="#">Schedule</a>	<a href="#">Enroll</a>
<b>Curriculum</b>	
ColdFusion	<a href="#">▶</a>
Dreamweaver	<a href="#">▶</a>
Sencha	<a href="#">▶</a>

Note: You can either use the static `getfaculty.json` file to populate the tree control or you can call `/ftst/components/subgenius.cfc?method=getfaculty` if your workstation is configured to run ColdFusion.

## Objectives

After completing this lab, you should be able to:

- Use Ext.List to output a contact list
- Use Ext.NestedList to output a list of courses
- Make an ad-hoc AJAX request to retrieve detailed information about a leaf node in a nested list

## Steps

### Review the Assets

1. Open `/lab/lab5/index.htm` in your editor and review its contents with your instructor. This is the solution to lab 4 with some additional CSS styles added.

2. Open the data file and review its structure:  
/ftst/json/getfaculty.json

### Part 1: Build the Contacts Page

3. Where indicated by the comment, define a model named **professor** that matches with the data returned from the URL that you reviewed in step 2.
4. Define a proxy for the model that reads data from the following URL via AJAX:

```
../../../../json/getfaculty.json
```

5. Review your code to ensure it is similar to the following:

```
Ext.regModel('professor', {
    fields: [
        {name: 'LNAME', type: 'string'},
        {name: 'FNAME', type: 'string'},
        {name: 'EMAIL', type: 'string'},
        {name: 'PHONE', type: 'string'},
        {name: 'BIO', type: 'string'},
        {name: 'SPECIALTY', type: 'string'}
    ],
    proxy: {
        type: 'ajax',
        url : '../../../../json/getfaculty.json',
        reader: {
            type: 'json'
        }
    },
});
```

6. Where indicated by the comment, define a variable named **profStore** that points to a new data store for the model that you defined in the previous step.
7. Add a configuration attribute to your Store that causes it to automatically load the dataset from the URL.
8. Review your code to ensure that it is similar to the following:

```
profStore = new Ext.data.Store({
    model: 'professor',
    autoLoad: true
});
```

9. Where indicated by the comment, delete the **contentEl** configuration attribute from the Panel definition
10. Within the same Panel definition, add the following attributes:
  - layout: {type: 'fit'}
  - items: [ ]
  - dockedItems: [ ]

11. Inside the items array that you defined in step 10, define an Ext.List with the following configuration properties:

- store: profStore
- itemTpl: '<div class="profcontact">{LNAME},{FNAME}<br />{SPECIALTY}<br />{EMAIL}<br />{PHONE}</div>'
- grouped: true
- indexBar: true
- pinHeaders: true
- clearSelectionOnDeactivate : true

#### **Enable Grouping on the Contacts Page**

12. Prior to your instantiation of the variable **profStore**, define a new variable named **profGroupBy** and assign it the string value of “**Lname**”
13. Add the following configuration attribute to the profStore declaration:

```
getGroupString : function(record) {
    if (profGroupBy == 'LNAME')
        return record.get('LNAME')[0];
    else
        return record.get('SPECIALTY')
}
```

14. Save the file and browse it. You should see the contacts list

#### **Enable a Group Toggle Buttons**

15. In the **dockedItems** array of your panel definition, define a toolbar docked at the top of the panel with a title of “Professors”
16. Define a button with a label of “By Name” on the toolbar that, when clicked, sets the **profGroupBy** variable to the string “**LNAME**” and sorts the data store by LNAME.

#### **Add a spacer to the toolbar**

17. Define a second button on the toolbar with a label of “By Specialty” that, when clicked, sets the **profGroupBy** variable to the string “**SPECIALTY**” and sorts the data store by SPECIALTY.

18. Save the file and test. Clicking on the group/sort buttons should change the order of the contacts list.

19. Review your toolbar definition to ensure it is similar to the following:

```
dockedItems: [
  {
    xtype: 'toolbar',
    dock: 'top',
    title: 'Professors',
    items: [
      {
        text: 'By Name',
        handler: function() {
          profGroupBy = "LNAME",
          profStore.sort("LNAME")
        }
      },
      {xtype: 'spacer'},
      {
        text: 'By Specialty',
        handler: function() {
          profGroupBy = "SPECIALTY";
          profStore.sort("SPECIALTY");
        }
      }
    ]
  }
]
```

## Part 2: Output the list of courses into a Nested List

20. Open the following file in your editor and review its structure:

```
/ftst/json/getclasses.json
```

21. Where indicated by the comment, define a Model named **coursesModel** that captures the information output from the URL that you reviewed in the prior step.

22. Review your code to ensure that it resembles the following:

```
Ext.regModel('coursesModel', {
  fields: [
    {name: 'text', type: 'string'},
    {name: 'courseid', type: 'int'}
  ],
  proxy: {
    type: 'ajax',
```

```

        url : '/json/getclasses.json',
        reader: {
            type: 'tree',
            root: 'items'
        }
    });
});

```

23. Immediately after your **coursesModel** definition, define a variable named **coursesStore** that points to a new **TreeStore** and uses the following configuration attributes:

- **model:** 'coursesModel'
- **storeId:** 'coursesStore'
- **autoLoad:** true

24. Review your **coursesStore** definition to ensure it resembles the following:

```

coursesStore = new Ext.data.TreeStore({
    model: 'coursesModel',
    storeId: 'coursesStore',
    autoLoad: true
});

```

25. Change your definition of coursesPnl from a Panel to a NestedList. Use the following configuration parameters:

- **id:** 'courselistning'
- **title:** 'Curriculum'
- **displayField:** 'text'
- **store:** coursesStore
- **fullscreen:** true
- **onItemDisclosure:** true
- **scroll:** 'vertical'

26. Define a **getDetailCard** attribute for your NestedList configuration that defines a function.

- Inside the function, display a message that data is loading
- Return a new Panel with the following properties
  - scroll: 'vertical'
  - layout: {type: 'fit'}
  - id: 'coursedetailpanel'

27. Review your CoursesPnl definition to ensure it resembles the following code:

```
var coursesPnl = new Ext.NestedList({
    id: 'courselist',
    title: 'Curriculum',
    displayField: 'text',
    store: coursesStore,
    fullscreen: true,
    onItemDisclosure: true,
    scroll: 'vertical',
    getDetailCard: function(record, parentRecord) {
        coursesPnl.setLoading(true);
        return new Ext.Panel(
            {
                scroll: 'vertical',
                layout: {type: 'fit'},
                id: 'coursedetailpanel'
            }
        );
    }
});
```

28. Save the file and browse. You should be able to click through the first two branches of the data structure.

#### Define a leafitemtap event handler

29. Immediately after your coursesPnl definition, define a leafitemtap event handler that makes a cross-domain request to a web service at following URL:

<http://senchatraining.com/ftst/components/subgenius.cfc>

Pass the following URL arguments to the request:

- method: 'getclassinfojsonp'
- courseid: r.get('courseid')

30. Verify your code to ensure it resembles the following:

```
coursesPnl.on('leafitemtap',
    function(subList, subIdx, el, e, detailCard) {
        var ds = subList.getStore(),
            r = ds.getAt(subIdx);
        Ext.util.JSONP.request({
            url:
                'http://senchatraining.com/ftst/components/subgenius.cfc',
            callbackKey: 'callback',
            params: {
                method: 'getclassinfojsonp',
                courseid: r.get('courseid')
            },
            callback: function(response) { }
        });
    });
});
```

31. Inside the callback handler for your JSON-P request.

- If the request was successful, turn off the “loading” message and output the COURSETEASER data from the resulting dataset.
- If the request was unsuccessful, turn off the “loading” message and output a message indicating that the system was unable to retrieve the data

32. Compare your code to the following:

```
callback: function(response) {
    if (response[0]) {
        dataObj = response[0];
        detailCard.update("<p id='coursedetailinfo'>" +
            dataObj.COURSETEASER + "</p>");

        coursesPnl.dockedItems.items[0].setTitle(
            dataObj.COURSENAME);

        coursesPnl.setLoading(false);
    } else {
        detailCard.update("Loading failed.");
        coursesPnl.setLoading(false);
    }
}
```

33. Save your file and test

### Challenge Exercise

34. Define the Schedule panel so that it appears similar to the screen shot below by pulling data from the following URL:

`.../json/getschedule.json`

The screenshot shows a mobile application interface for 'SubGenius University'. The top navigation bar has four buttons: 'Welcome', 'Courses', 'Schedule', and 'Enroll'. The main content area displays three course schedule entries, each with a title, date, location, and instructor. The first entry is: #1: Administering ColdFusion : 12/01/2010 Washington, DC Steve Drucker. The second entry is: #2: Website Development : 12/05/2010 Washington, DC David Gallerizzo. The third entry is: #3: Sencha Touch : 12/15/2010 San Francisco, CA David Watts. At the bottom of the screen are two buttons: 'By Date' and 'By Name'.

Illustration 7: Can you create this without further instructions?

– End of Lab --

---

---

# **Unit 6:**

## **Implementing Advanced GUIs**

### **Unit Objectives**

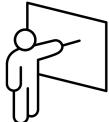
After completing this unit, you should be able to:

- Add multimedia to your application
- Deploy a Google Map
- Deploy charts in order to spot trends in your data sets

### **Unit Topics**

- Adding Audio and Video to your Applications
- Adding Google Maps to your Applications
- Visualizing Data with Charts

# Adding Audio and Video to your Applications



You will use the following Sencha Touch classes to embed multimedia within your application:

- `Ext.Media` is a base class for audio/video controls and should not be used directly
- `Ext.Audio` provides a simple wrapper for HTML 5 audio
- `Ext.Video` Provides a simple container for HTML 5 video

## Implementing Media

The `Ext.Media` class extends `Ext.Container` by providing the following configuration attributes:

Config Option	Description
<code>autoPause</code>	Boolean. Will automatically pause the media when the container is deactivated. (Defaults to true)
<code>autoResume</code>	Boolean. Will automatically start playing the media when the container is activated. (Defaults to false)
<code>enableControls</code>	Boolean. Set this to false to turn off the native media controls (Defaults to true).
<code>preload</code>	Boolean. Immediately begins downloading the media
<code>url</code>	String. Location of the Media to play.

`Ext.Media` also supports the following methods which you can use to develop custom controls:

Method	Description
<code>pause()</code>	Pauses playback
<code>play()</code>	Starts playback, or continues playback from the paused position
<code>toggle()</code>	Toggles the media playback state

## Deploying Audio

The Ext.Audio class provides a simple container for HTML 5 audio.

Recommended file formats include:

- Uncompressed WAV audio
- Uncompressed AIF audio
- MP3 audio
- AAC-LC audio
- HE-ACC audio

The following example illustrates playback of an mp3 audio file:

```
var pnl = new Ext.Panel({  
    fullscreen: true,  
    items: [{  
        xtype: 'audio',  
        url: "who-goingmobile.mp3"  
    }]  
});
```

Ext.Audio does not have any class-specific configuration properties.

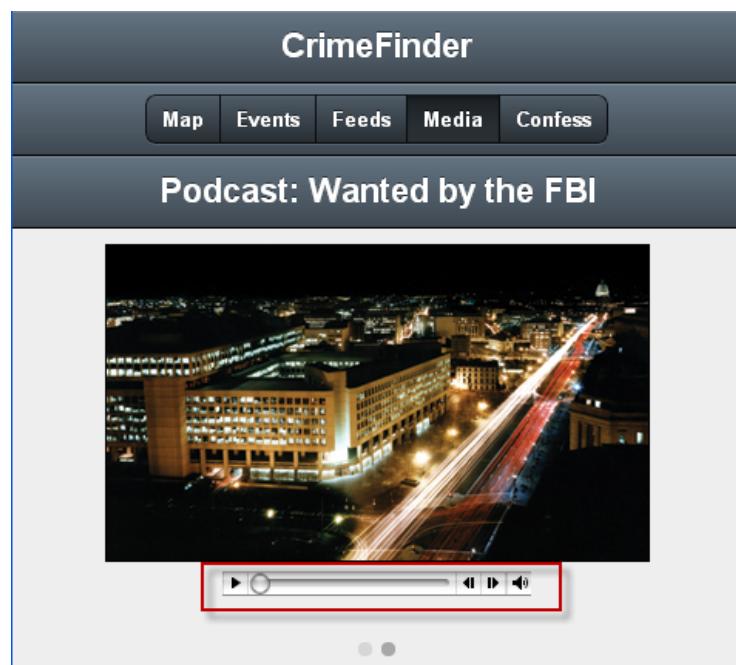


Illustration 1: Audio controls are highlighted by the red box

## Deploying Video

The Ext.Video class provides a simple container for HTML 5 video.

Recommended file formats include:

- H.264 format
- .mov format

The following example illustrates playback of a video file:

```
var pnl = new Ext.Panel({  
    fullscreen: true,  
    items: [{  
        floating: true,  
        x: 600,  
        y: 300,  
        width: 175,  
        height: 98,  
        xtype: 'video',  
        url: "porsche911.mov",  
        posterUrl: 'porsche.png'  
    }]  
});
```

Ext.Video supports a single class-specific configuration option – posterURL, which is the location of a poster image to be shown prior to video playback.



Illustration 2: HTML 5 Video Playback in Sencha Touch

## Dynamically Instantiating Panels

You may be tasked with conditionally instantiating panels based on your business rules. For instance, you might have a data set where only a certain number of records refer to a video clip. In this use case, you might want to alternate displaying text-based information with video if it is available.

Use the Ext.Panel.add() method to deploy a nested panel as indicated by the following code sample:

```
if (myData.VIDEOINTROURL == '') {  
    myPanel.update("No Video Available");  
} else {  
    myPanel.add({  
        xtype: 'video',  
        url: myData.VIDEOINTROURL,  
        posterUrl: myData.VIDEOINTROPOSTER  
    });  
    myPanel.doLayout();  
}
```

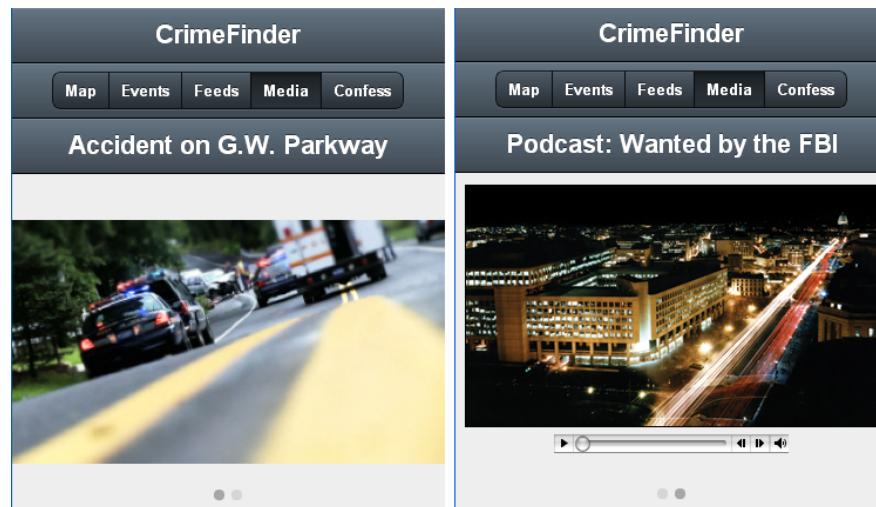
*Note that you must call the Ext.Panel.doLayout() method to refresh a panel's contents and display dynamically instantiated children elements.*

## Walkthrough 6-1: Integrating Audio and Video



In this walkthrough, you will perform the following tasks:

- Play a video from within your application
- Play an MP3 file from within your application
- Display MP3 controls underneath a JPG image



**Illustration 3: The video and audio that you will deploy during this exercise**

## Steps

### Review the Starting Code and Data

1. Using your editor, open [/walk/walk6-1/index.htm](#). Note that this is essentially the solution from the prior walkthrough.

### Add the Video

2. Where indicated by the comment, define a panel with the following attributes:
  - xtype: 'panel'
  - layout: {type: 'card'}
3. Verify that your code appears as follows:

```
{xtype: 'panel', cls: 'card1', layout: 'card'}
```
4. Add an items array to the panel you defined in the prior step.

5. In the items array, define a video with the following attributes:
  - fullscreen: true
  - xtype: 'video'
  - url: '../..//video/accident.mp4'
  - posterUrl: '../..//video/poster1.png'
6. Add a dockedItems array to the panel that you defined in step 3.
7. Define a single docked item with the following attributes:
  - xtype: 'toolbar'
  - dock: 'top'
  - title: 'Accident on G.W. Parkway'
8. Save the file and browse
9. Verify that your code resembles the following:

```
{
  xtype: 'panel',
  cls: 'card1',
  layout: {type: 'card'},
  items: [
    {
      fullscreen: true,
      xtype: 'video',
      url: '../..//video/accident.mp4',
      posterUrl: '../..//video/poster1.png',
    }],
  dockedItems: [
    {
      xtype: 'toolbar',
      dock: 'top',
      title: 'Accident on G.W. Parkway'
    }
  ]
}
```

### Add the Audio

10. Where indicated by the comment, define a panel with the following options:
  - xtype: 'panel'
  - layout: {type: 'vbox'}
  - items: [ ]
  - dockedItems: [ ]
11. Inside the items array that you defined in the previous step, define a panel that outputs an <img> tag pointing to ../..//images/fbi.png.

12. Verify that your panel definition resembles the following:

```
{  
    xtype: 'panel',  
    layout: {type: 'vbox'},  
    items: [  
        {  
            xtype: 'panel',  
            html: ''  
        }  
    ],  
    dockedItems: []  
}
```

13. Add the audio instantiation to the items array depicted in step 12. The audio clip is at the following url:

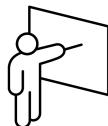
**`.../.../video/podcast.mp3`**

14. Define a toolbar in the dockedItems array that has the title “Podcast: Wanted by the FBI”
15. Save the file and test. Your media panel should resemble the screenshots on page 6-5
16. Verify that your complete videoPnl definition resembles the code listing on the next page.

```
var videoPnl = new Ext.Carousel({
    direction: 'horizontal',
    items: [
        /* step 2 */
        {
            xtype: 'panel',
            cls: 'card1',
            layout: {type: 'card'},
            items: [{{
                fullscreen: true,
                xtype: 'video',
                url: '../../video/accident.mp4',
                posterUrl: '../../video/poster1.png',
            }}],
            dockedItems: [
                {
                    xtype: 'toolbar',
                    dock: 'top',
                    title: 'Accident on G.W. Parkway'
                }
            ]
        },
        // step 10
        {
            xtype: 'panel',
            layout: {type: 'vbox'},
            items: [
                {
                    xtype: 'panel',
                    html: '',
                    },
                    {
                        xtype: 'audio',
                        url: '../../video/podcast.mp3'
                    }
                ],
                dockedItems: [
                    {
                        xtype: 'toolbar',
                        title: 'Podcast: Wanted by the FBI'
                    }
                ]
            }
        }
    );
});
```

-- End of Walkthrough --

## Adding Google Maps to your Applications



The Ext.Map class wraps a Google map in an Ext.Component. To use the component you must include the following JavaScript in your page:

```
<script type="text/javascript"
       src="http://maps.google.com/maps/api/js?sensor=true">
</script>
```

The following example illustrates a simple map instantiation:

```
var pnl = new Ext.Panel({
    fullscreen: true,
    items: [{
        xtype: 'map',
        useCurrentLocation: true
    }]
});
```

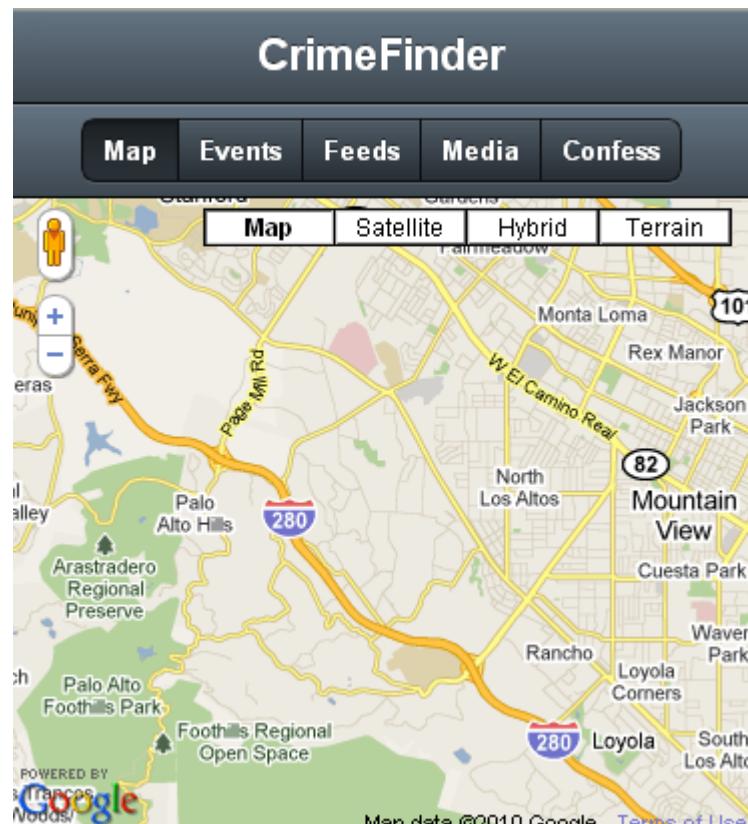


Illustration 4: A simpe instantiation of a Google Map

## Configuring the Map

The following configuration attributes are specific to Ext.Map:

Config Option	Description
baseCls	String. The base CSS class to apply to the Map element (defaults to 'x-map')
mapOptions	Object. Described in the Google Documentation at <a href="http://code.google.com/apis/maps/documentation/v3/reference.html">http://code.google.com/apis/maps/documentation/v3/reference.html</a>
maskMap	Boolean. Masks the map (defaults to false)
maskMapCls	String. CSS class to add to the map when maskMap is true.
useCurrentLocation	Boolean. Pass in true to center the map based on geolocation coordinates.

## Configuring Google Map Options

Configurable Google Map options include the following:

Map Option	Description
center	Centers the map at a specific latitude and longitude
zoom	Integer. Zoom level
mapTypeId	The initial mapTypeId. Options include HYBRID, ROADMAP, SATELLITE, and TERRAIN
mapTypeControl	Boolean. The initial enabled/disabled state of the map type control
navigationControl	Boolean. The initial enabled/disabled state of the navigation control.
navigationControlOptions	The initial display options for your navigational controls.

The following example displays a map that is centered at 1400 16<sup>th</sup> Street NW, Washington DC:

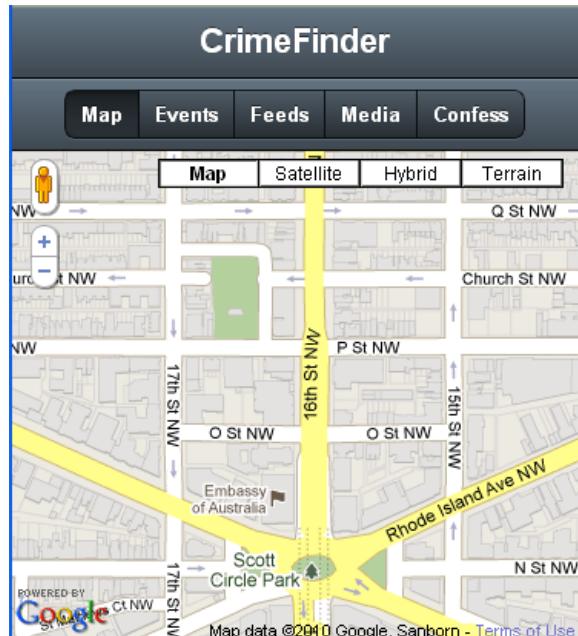


Illustration 5: Map centered at 1400 16th st NW

```
{
    xtype: 'map',
    useCurrentLocation: false,
    mapOptions : {
        center : new google.maps.LatLng(38.909085,-77.036777),
        zoom : 16,
        mapTypeId : google.maps.MapTypeId.ROADMAP,
        navigationControl: true,
        navigationControlOptions: {
            style: google.maps.NavigationControlStyle.DEFAULT
        }
    }
}
```

## Determining the user's current location

Use the `navigator.geolocation.getCurrentPosition()` method to return the physical location of the device. The method executes asynchronously, invoking a function when it has determined your position. You can subsequently extract the latitude/longitude coordinates and translate them into a Google point as illustrated by the following example:

```
navigator.geolocation.getCurrentPosition(function(position) {
    var initialLocation = new
        google.maps.LatLng(position.coords.latitude,
                           position.coords.longitude);
})
```

## Geocoding an Address

In certain circumstances you may want to center the map on an address other than the user's current location. In order to accomplish this you will need to translate an address into a latitude/longitude through a process referred to as "geocoding"

You can geocode an address using the `google.maps.Geocoder()` class as indicated by the following code sample.

Note the following:

- The geocoding service executes asynchronously, so you need to specify a handler function to interpret the results.
- The geocoding service returns a results array and a status message. If the geocoding operation was successful, status will be equal to the constant `google.maps.GeocoderStatus.OK`.
- The location of the address as a latitude,longitude point is available through `results[0].geometry.location`
- You can programatically center the map on a specific coordinate using the `Ext.Map.update()` method.

```
gcAddress = function(address) {
    var geocoder = new google.maps.Geocoder();
    geocoder.geocode({'address': address},
        function(results, status) {
            if (status != google.maps.GeocoderStatus.OK) {
                Ext.Msg.alert("Address not found", status);
            } else {
                mapCenter = results[0].geometry.location;
                var extMap = Ext.getCmp('googleMap');
                extMap.update(mapCenter);
            }
        })
}
```

## Adding Markers to your Map

Use the `google.maps.Marker` constructor to add a marker to your map. You can specify a configuration object with the following attributes:

Config Option	Description
<code>position</code>	(Required) A google latitude/longitude point
<code>map</code>	(Required) A pointer to the Google map
<code>title</code>	A string to display as a tooltip for the marker

The following example illustrates adding a marker to a map:

```
var marker = new google.maps.Marker(
{
  position: new google.maps.LatLng(37.0625,-95.67706),
  map: Ext.getCmp('googleMap').map,
  title: results[0].formatted_address
}
);
```

*Note: Other configuration options are available, including the ability to specify your own custom icons. Consult the Google Maps API documentation for more details.*

## Displaying an Information Window

Use the `google.maps.InfoWindow()` class to define an information “bubble” containing text as illustrated by the following screen shot. Note that you will also need to add an event listener to the map marker so that clicking it causes the info window to display.

The following code sample illustrates defining a marker, info window, and event listener. Note that you could use the marker 'click' event to also invoke methods within Sencha Touch that could populate a panel with data and set focus to it.



Illustration 6: A Google Map Info Window

```
var infoWindow = new google.maps.InfoWindow({
  content: 'This is where I work'
});
var marker = new google.maps.Marker(
{
  position: new google.maps.LatLng(37.0625,-95.67706),
  map: Ext.getCmp('googleMap').map,
  title: 'My office'
}
);

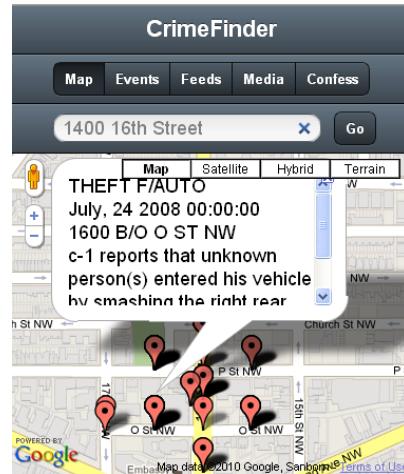
google.maps.event.addListener(marker,'click', function() {
  infoWindow.open(Ext.getCmp('googleMap').map, marker);
})
```

## Walkthrough 6-2: Implementing a Google Map



In this walkthrough, you will perform the following tasks:

- Display a Google Map
- Center the map on a specific address
- Make an AJAX call to retrieve crimes that occurred within a 0.1 mile radius of the address
- Mark points on the map where crimes occurred
- Add a click handler to a map point that displays detailed information about the crime in an info window



**Illustration 7: You will plot points on a Google map during this walkthrough**

### Steps

#### Review the Starting Code

1. Using your editor, open `/walk/walk6-2/index.htm`. Note that this is essentially the solution from the prior walkthrough.
2. Remove the `html` attribute from the `mapPnl` panel configuration

#### Instantiate the Map

3. Add an `items` array to the `MapPnl` configuration object
4. Inside the `items` array that you defined in the prior step, define a map panel using the following attributes:
  - `id: googleMap`
  - `xtype: 'map'`
  - `useCurrentLocation: false`
5. Save the file and browse. You should see a map appear.
6. Return to your editor

7. Add a **mapOptions** property to your map definition. Use the following attributes:
  - center: new google.maps.LatLng(38.909085,-77.036777)
  - zoom: 16
  - mapTypeId: google.maps.MapTypeId.ROADMAP
  - navigationControl: true
  - navigationControlOptions: {  
          style: google.maps.NavigationControlStyle.DEFAULT  
        }
8. Save the file and test. You should see the map, centered on a location in Washington, DC
9. Verify that your code resembles the following:

```
var mapPnl = new Ext.Panel ({  
    items: [  
        {  
            id: 'googleMap',  
            xtype: 'map',  
            useCurrentLocation: false,  
            mapOptions : {  
                center : new google.maps.LatLng(38.909085,-77.036777),  
                zoom : 16,  
                mapTypeId : google.maps.MapTypeId.ROADMAP,  
                navigationControl: true,  
                navigationControlOptions: {  
                    style: google.maps.NavigationControlStyle.DEFAULT  
                }  
            }  
        }  
    ]  
})
```

### Add the Address Search Box

10. Add a **dockedItems** attribute to your **mapPnl** definition
11. Inside the **dockedItems** array, define a toolbar that is docked at the top of the panel.
12. Define the following elements for the toolbar (listed in order):
  - A search field with the following properties:
    - id: 'searchAddress'
    - name: 'searchAddress'
    - xtype: 'searchfield'
    - width: '60%'
    - placeHolder: 'DC Street Address'
  - A button with the following attributes:
    - xtype: 'button'
    - text: 'Go'
    - handler: function() {}
13. Center the buttons on the toolbar
14. Save the file and browse it. You should see your toolbar docked at the top of your map.
15. Ensure your code resembles the following:

```
dockedItems: [
  {
    xtype: 'toolbar',
    layout: {pack: 'center'},
    items: [
      {
        id: 'searchAddress',
        xtype: 'searchfield',
        name: 'searchAddress',
        width: '70%',
        placeHolder: 'DC Street Address'
      },
      {
        xtype: 'button',
        text: 'Go',
        handler: function() {}
      }
    ]
}]
```

### Geocode the Address

16. Inside the button handler function that you defined in step 12, declare a local variable named **s** that concatenates the value of the **searchAddress** field with “ Washington, DC”. Your code should appear as follows:

```
var s = Ext.getCmp('searchAddress').getValue() +  
        " Washington, DC";
```

17. Directly underneath the code that you inserted in the prior step, invoke a function named **geoCodeAddress**, passing **s** as a parameter.
18. Where indicated by the comment, copy and paste the code located in **/snippets/walk6-2a.js**. Review the code with your instructor.
19. Where indicated by the comment, define a function named **plotLocation** that accepts two arguments named **pos** and **address**
20. Inside the **plotLocation** function, define a local variable named **extMap** that points to your Google Map component:

```
var extMap = Ext.getCmp('googleMap');
```

21. Immediately after the code that you inserted in the prior step, center the map on the position passed into the function.

```
extMap.update(pos);
```

22. After centering the map, deploy a map marker at the position passed into the function as indicated below:

```
var marker = new google.maps.Marker({  
    position: pos,  
    map: extMap.map,  
    title: 'The location'  
});
```

23. Save the file and browse in Safari
24. Input the following address into the search box:

**2000 K st NW**

25. Press the **Go** button
26. The map should center on the address that you typed and you should see a map marker.

### Make an AJAX request for crimes in the vicinity

27. Directly underneath the code that you inserted in step 27, invoke a function named **loadCrimeData()** passing the position and address as arguments:

```
loadCrimeData(pos, address);
```

28. Where indicated by the comment, define a function named **loadCrimeData()** that accepts two arguments – **pos** and **address**.
29. Inside the **loadCrimeData()** function, reconfigure the proxy of the crimeStore object to point at the following url:

```
'http://www.senchatouchtraining.com/ftst/components/' +
'crimeservice.cfc?method=crimesearchjsonp&address=' +
escape(address) + '&pos=' + pos.lat() + ',' + pos.lng()
```

30. Verify that your function appears as follows:

```
loadCrimeData = function(address) {
    crimeStore.setProxy ({
        type: 'scripttag',
        url: 'http://www.senchatouchtraining.com/ftst/components/crimeservice.cfc?
method=crimesearchjsonp&address=' + escape(address) + '&pos=' + pos.lat() + ','
+ pos.lng()
    });
}
```

31. Invoke the crimeStore.load() method, defining a callback function that loops through each of the resulting records as indicated by the following:

```
crimeStore.load({
    scope: this,
    callback: function(records, operation) {
        Ext.each(records, function(record) {
            })
    }
});
```

32. Inside the **Ext.each()** callback function, define a google maps info window that outputs the **OFFENSE**, **REPORTDATE**, **ADDRESS**, and **DESCRIPTION** fields from each record. Your code should appear similar to the following:

```
var infoWindow = new google.maps.InfoWindow(
{
    content: record.get('OFFENSE') + '<br>' +
        record.get('REPORTDATE') + '<br>' +
        record.get('ADDRESS') + '<br>' +
        record.get('DESCRIPTION')
})
);
```

33. Immediately after the code you inserted in the prior step, define a Google Map Marker at the latitude/longitude position returned from the web service. Your code should appear as follows:

```
var marker = new google.maps.Marker({  
    position: new google.maps.LatLng(  
        record.get('LAT'), record.get('LNG')),  
    map: Ext.getCmp('googleMap').map,  
    title: record.get('OFFENSE')  
});
```

34. Add an event listener to the map marker that displays the **infoWindow** if it is pressed:

```
google.maps.event.addListener(marker, 'click', function() {  
    infoWindow.open(Ext.getCmp('googleMap').map, marker);  
})
```

35. Save the file and test. Your complete **loadCrimeData** function should appear as follows:

```
loadCrimeData = function(address) {  
    crimeStore.setProxy (  
    {  
        type: 'scripttag',  
        url:  
        'http://www.senchatouchtraining.com/ftst/components/crimeservice.cfc?method=crimesearchjsonp&address=' + escape(address)  
        + '&pos=' + pos.lat() + ',' + pos.lng(),  
  
        reader: { type: 'json'}  
    });  
  
    crimeStore.load(  
    {  
        scope: this,  
        callback: function(records, operation)  
        {  
            Ext.each(records, function(record)  
            {  
                var infoWindow = new google.maps.InfoWindow(  
                {  
                    content: record.get('OFFENSE') + '<br>' +  
                    record.get('REPORTDATE') + '<br>' + record.get('ADDRESS')  
                    + '<br>' + record.get('DESCRIPTION')  
                }  
                );  
                var marker = new google.maps.Marker({  
                    position: new google.maps.LatLng(record.get('LAT'),  
                        record.get('LNG')),  
                    map: Ext.getCmp('googleMap').map,  
                    title: record.get('OFFENSE')  
                });  
  
                google.maps.event.addListener(marker, 'click', function() {  
                    infoWindow.open(Ext.getCmp('googleMap').map, marker);  
                })  
            }  
        }  
    });  
}
```

Note that by changing the contents of the **crimeStore**, you also simultaneously changed the contents of the Events panel.

#### Add a button to plot the user's current position

36. Define an additional button on your Map panel toolbar using the following config attributes:

- xtype: 'button'
- iconCls: 'locate'
- iconMask: true
- handler: {}

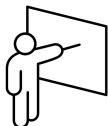
37. Inside the button handler, use the **navigator.geolocation.getCurrentPosition()** method to get the user's current position and plot it on the map. Your code should appear as follows:

```
navigator.geolocation.getCurrentPosition(function(position) {  
  
    var initialLocation = new  
        google.maps.LatLng(position.coords.latitude,  
                           position.coords.longitude);  
  
    plotLocation(initialLocation,'');  
}  
)
```

38. Save the file and test.

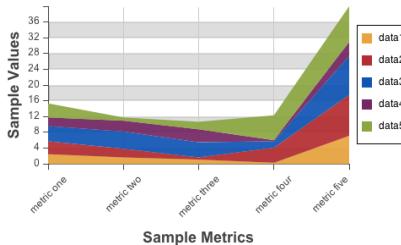
– End of Walkthrough –

# Visualizing Data with Charts

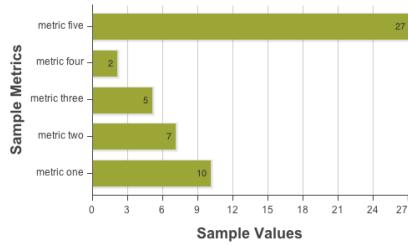


Sencha Touch Charts features beautiful, hardware accelerated HTML5 charts that let you visualize complex data sets in an easy to use gesture based interface on Apple, Android, and Blackberry devices.

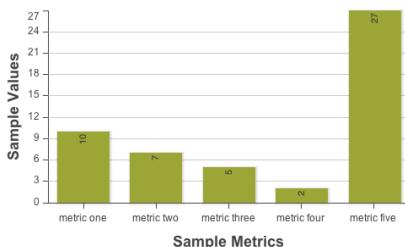
An abstraction of the HTML5 <canvas> tag, Sencha Touch Charts is comprised of a low-level sprite-based drawing API and a high-level charting API that supports the following types of data visualizations:



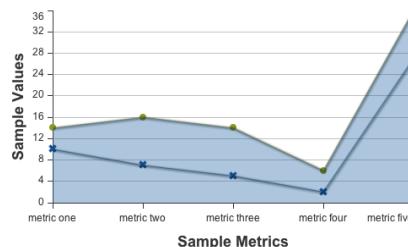
Area



Bar



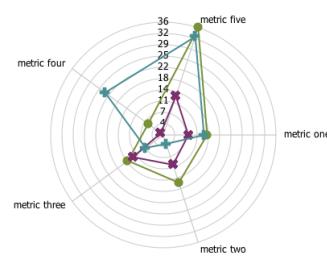
Column



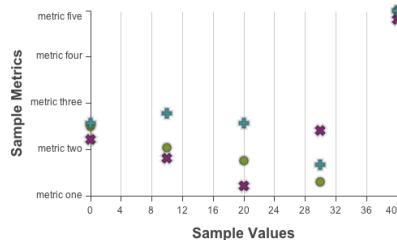
Line



Pie



Radar



Scatter



Gauge

## Licensing Sencha Touch Charts

Sencha Touch Charts 1.0 is available under the GPLv3 for open source projects, which may require the release of your code. You may also purchase a commercial license. The license you choose for Sencha Touch Charts must match the license with which you use Sencha Touch.

## Getting Started with Sencha Touch Charts

Sencha Touch Charts ships as a .ZIP file and should unpacked into a web-accessible directory. Enabling charting requires you to include a <script> tag within your HTML that links to either the touch-charts-debug.js file, or its minified version, touch-charts.js.

The charting framework also requires that you use Sencha Touch version 1.1.1 or later which fixes several issues with MultiTouch detection and orientation problems on non-iOS devices. Sencha Touch 1.1.1 (sencha-touch.js and sencha-touch-debug.js) are present in the Sencha Touch Charts distribution. You must also load a CSS file that contains updated style definitions for charting.

Typical implementations will therefore contain code resembling the following snippet:

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

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

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

## Calculating Aggregate Functions on the Client

Sencha Touch Charts visualize data from a Store. You can easily calculate client-side aggregate functions by extending the Ext.data.Store class to include an embedded store to contain statistical data as indicated by the following code sample.

```
var CountStore = Ext.extend(Ext.data.Store, {

    /* define embedded store */
    countStore: new Ext.data.JsonStore({
        fields: ['label', 'count']
    }),

    /* define getter */
    getCount: function(){
        return this.countStore;
    },

    /* generate count(*) statistics */
    calcCount: function(){
        var aGroups = this.getGroups();
        var aResult = [];
        for (var i = 0; i < aGroups.length; i++) {
            aResult.push({
                label: aGroups[i].name,
                count: aGroups[i].children.length
            });
        }
        this.countStore.loadData(aResult);
        return aResult;
    },

    /* recalc stats on load */
    listeners: {
        load: {
            fn: function(objStore, aRecords, bSuccess){
                this.calcCount();
            }
        }
    }
});

/* create store instance */

var crimeStore = new CountStore({
    storeId: 'crimeStore',
    model: 'CrimeReport',
    autoLoad: true,
    remoteFilter: true,
    remoteSort: true,
    groupField: 'OFFENSE'
});
```

## Instantiating a Chart

The Ext.chart.Panel constructor gives you the capability to visualize data by graphically representing data from a Ext.data.Store. Chart configuration objects can typically be broken down into the following sections:

- Chart Properties
- Chart Legend (if applicable)
- Chart Axes
- Chart Interactions
- Chart Series

Note that charts may only access data from an Ext.data.Store.

Charts may be themed using the following properties:

Property	Description
animate	Boolean Object. <code>true</code> for the default animation (easing: 'ease' and duration: 500) or a standard animation config object to be used for default chart animations. Defaults to false.
background	Boolean Object. Set the chart background. This can be a gradient object, image, or color. Defaults to false for no background.
theme	String. The name of the theme to be used. A theme defines the colors and other visual displays of tick marks on axis, text, title text, line colors, marker colors and styles. Possible theme values are 'Base', 'Green', 'Sky', 'Red', 'Purple', 'Blue', 'Yellow' and also six category themes 'Category1' to 'Category6'. Default value is 'Base'.

Typical chart configuration objects usually contain the following structure:

```
new Ext.chart.Panel ({
    title: 'My Chart',
    items: [
        store: store1,
        animate: true,
        shadow: true,
        theme: 'Category1',
        legend: { position: 'right' },
        axes: [ ...some optional axes options... ],
        interactions: [ ... some optional interactions ...],
        series: [ ...some series options... ]
    ]
})
```

## Defining the Chart Legend

Configuration properties for your chart's legend include the following:

Config Property	Description
dock	Boolean. If set to <code>true</code> , the legend will be docked to the configured edge position within an Ext.Sheet.
doubleTapThreshold	The duration in milliseconds in which two consecutive taps will be considered a doubletap. Defaults to 250.
position	The position of the legend in relation to the chart. Valid values include 'top', 'bottom', 'left' and 'right'. You can also specify an Object with numeric properties <code>x</code> and <code>y</code> , and the boolean property <code>vertical</code> to display the legend floating on top of the chart at the given x/y coordinates.
visible	Boolean. Whether or not the legend should be displayed.

## Setting the Legend Position

You can specify different legend alignments based on the orientation of the browser viewport. For instance, you might want to put the legend on the right in landscape orientation but on the bottom in portrait orientation. To achieve this, you can set the `position` config to an Object with `portrait` and `landscape` properties, and set the value of those properties to one of the recognized value types described above. For example, the following config will put the legend on the right in landscape but float it on top of the chart at position 10,10 in portrait:

```
legend: {
    position: {
        landscape: 'right',
        portrait: {
            x: 10,
            y: 10,
            vertical: true
        }
    }
}
```

## Setting the dock Property

Setting the legend's dock property to true results in the the sheet will be initially hidden and can be opened by tapping on a tab along the configured edge. This prevents screen real estate from being taken up by the legend, which is especially important on small screen devices. Defaults to `true` for phone-sized screens, `false` for larger screens.

Note that the chart's legend will automatically dock on small screen devices, as illustrated by the following screen shots:

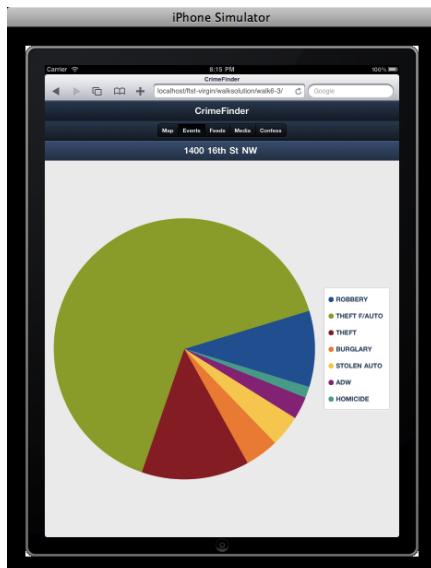


Illustration 8: iPad Chart



Illustration 9: iPhone Chart

## Defining Axes

Axes represent the types of data that you are charting. The axis package contains an Abstract axis class that is extended by Axis and Radial axes. Axis represents a Cartesian axis and Radial uses polar coordinates to represent the information for polar based visualizations like Pie and Radar series. The axis position, type, style can be configured. The axes are defined in an axes array of configuration objects where the type, field, grid and other configuration options can be set.

Five types of Axes are available:

- `category` representing categorical information
- `numeric` representing quantitative information
- `radial` representing a circular display of numerical data by steps
- `time` representing data that occurs over a period of time
- `gauge` enabling you to label positions on an arc

Basic configuration properties for Axes include the following:

Property	Description
<code>fields</code>	Array. An array containing the names of record fields which should be mapped along the axis
<code>label</code>	String. The label configuration object for the Axis
<code>position</code>	Where to set the axis. Available options are left, bottom, right, top. Default's bottom.
<code>title</code>	The title for the axis

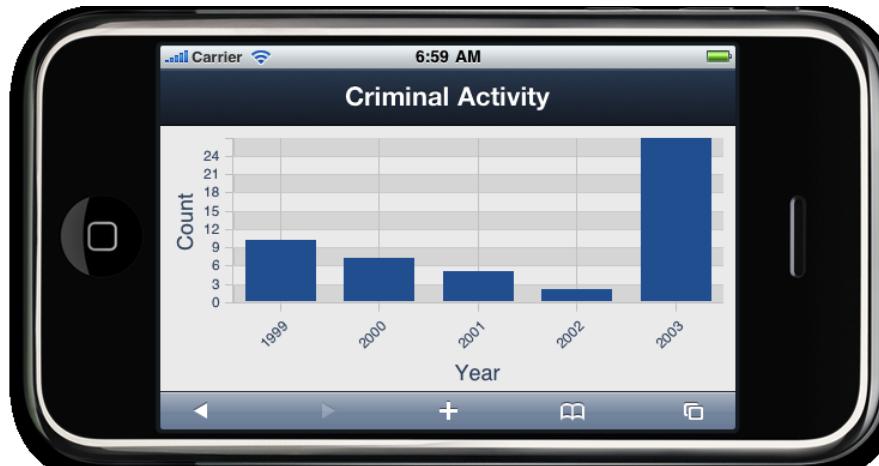


Illustration 10: Numeric and Category Axes

The chart axes depicted in Illustration 10 were generated by the following definition:

```
axes: [{  
    type: 'Numeric',  
    position: 'left',  
    fields: ['data1'],  
    title: 'Count',  
    grid: {  
        odd: {  
            opacity: 1,  
            fill: '#ddd',  
            stroke: '#bbb',  
            stroke-width: 1  
        }  
    },  
    minimum: 0,  
    adjustMinimumByMajorUnit: false  
}, {  
    type: 'Category',  
    position: 'bottom',  
    fields: ['name'],  
    title: 'Year',  
    grid: true,  
    label: {  
        rotate: {  
            degrees: 315  
        }  
    }  
}]
```

## Defining Interactions

The Ext.chart.interactions package contains a number of classes that make your charts more interactive.

Class	Description
ItemCompare	Allows the user to select two data points in a series and see a trend comparison between the two. An arrowed line will be drawn between the two points.
ItemHighlight	Allows highlighting of series data items on the chart.
ItemInfo	Allows displaying detailed information about a series data point in a popup panel.
PanZoom	Allows the user to navigate the data for one or more chart axes by panning and/or zooming. Navigation can be limited to particular axes. Zooming is performed by pinching on the chart or axis area; panning is performed by single-touch dragging.
PieGrouping	Allows the user to select a group of consecutive slices in a pie series to get additional information about the group. It provides an interactive user interface with handles that can be dragged around the pie to add/remove slices in the selection group.
Reset	Allows resetting of all previous user interactions with the chart. By default the reset is triggered by a double-tap on the empty chart area;
Rotate	Allows rotation of a Pie or Radar chart series. By default rotation is performed via a single-finger drag around the center of the series, but can be configured to use a two-finger pinch-rotate gesture by setting gesture: 'pinch'.
ToggleStacked	Allows toggling a bar or column series between stacked and grouped orientations for multiple yField values. By default this is triggered via a swipeevent

These items are invoked through the interactions configuration property of Ext.chart.Chart as illustrated below. Note that some interactions require additional configuration.

```
interactions: [
  { type: 'reset' }
  { type: 'rotate' }
]
```

## Defining Series

A `Series` is an abstract class extended by concrete visualizations like `Line`, `Pie`, and `Column`. `Series` contains code such as event handling, animation handling, shadows, gradients, and common offsets. It is also enhanced with a set of mixins that enables labels, highlighting, callouts, and tool tips.

Subclasses of `Series` include the following:

- `Area`
- `Bar`
- `Column`
- `Scatter`
- `Line`
- `Pie`
- `Radar`
- `Gauge`

*Note: This course will only cover Pie charts.*

## Outputting Pie Charts

Pie Charts, represented by `Ext.chart.series.Pie` are useful in displaying quantitative information for different categories that also have a meaning as a whole.

You will typically set the following class-specific properties when defining a pie chart:

Property	Description
<code>colorSet</code>	An array of color values which will be used, in order, as the pie slice fill colors.
<code>donut</code>	Boolean Number. Can be set to a particular percentage to set the radius of the donut chart.
<code>field</code>	String. Required. The store record field name to be used for the pie angles. The values bound to this field name must be positive real numbers.
<code>lengthField</code>	The store record field name to be used for the pie slice lengths. The values bound to this field name must be positive real numbers. This parameter is optional.
<code>showInLegend</code>	Boolean. Whether to add the pie chart elements as legend items.
<code>title</code>	String. The human readable name for the Series

The following example illustrates an instantiation of a Pie chart:

```
countStore: new Ext.data.JsonStore({
    fields: ['label', 'count']
});

/* code to populate store omitted for brevity */

new Ext.chart.Panel({
    title: 'Sample Chart',
    id: 'chartPanel',
    items: [{
        store: countStore,
        shadow: false,
        animate: true,
        insetPadding: 20,
        legend: {
            position: {
                portrait: 'bottom',
                landscape: 'left'
            }
        },
        series: [
            {
                type: 'pie',
                field: 'count',
                showInLegend: true,
                label: {
                    field: 'label',
                    display: 'rotate',
                    contrast: true,
                    font: '12px Arial'
                }
            }
        ]
    }]
})
```

## Walkthrough 6-3: Working with Charts



In this walkthrough, you will perform the following tasks:

- Instantiate a pie chart
- Make the chart interactive
- Programmatically change the title of a Chart panel's toolbar.

### Steps

#### Review the Starting Code

1. Using your editor, open **/walk/walk6-3/index.htm**. Note that this is essentially the solution from the prior walkthrough.

#### Load Sencha Touch Charts

2. Modify the <script> tag that currently points to `/senchatouch/sencha-touch-debug.js` to instead point to `/touch-charts/sencha-touch-debug.js`
3. Where indicated by the comment, insert a <script> tag that points to `/touch-charts/touch-charts-debug.js`
4. Change the <link> tag to point at `/touch-charts/resources/css/touch-charts-demo.css`

#### Review the Data Store Definitions

5. Review the CountStore and CrimeStore definitions with your instructor.

#### Instantiate the Chart

6. Where indicated by the comment, define a new Ext.chart.Panel with the following attributes:
  - id: 'chartPanel'
  - items: []
  - title: 'Analysis'



**Illustration 11: Using a pie chart to visualize the distribution of crimes in a neighborhood**

7. Inside the items array that you defined in the previous step, define the following properties:
  - store: crimeStore.getCount()
  - shadow: false
  - animate: true
  - insetPadding: 20
  - legend: { }
  - series: [ ]
  - interactions: [ ]

#### Define the Chart Legend

8. Configure the legend property that you entered in the previous step as follows:
  - In portrait orientation, the legend should appear at the bottom of the panel.
  - In landscape orientation, the legend should appear to the left of the chart series.
9. Verify that your code appears similar to the following:

```
legend: {
    position: {
        portrait: 'bottom',
        landscape: 'left'
    }
}
```

#### Define the Chart Series

10. Configure the series property that you entered in step 7 to output a Pie chart by referencing the following attributes:
  - type: 'pie'
  - field: 'count'
  - showInLegend: true
11. Save the file and browse . Note that you can access the chart by swiping the Events list to the left.

### Add Labels to the Chart

12. Add labels to each of the pie slices by inserting a `label` attribute into the `pie` series definition. Experiment with changing values for the following attributes:
- `field: 'label'`
  - `display: 'rotate'`
  - `contrast: true`
  - `font: '12px Arial'`

### Enable Chart Interactions

13. Enable chart rotation by adding the following object to the interactions array that you defined in step 7.

```
{ type: 'rotate' }
```

14. Enable the user to get detailed information an item in the store by tapping and holding its representative pie slice. Add the following object to the interactions array:

```
{type: 'iteminfo', gesture: 'taphold'}
```

15. Add a 'show' event listener to the code that you inserted in the previous step. Inside the handler, output the contents of the `label` and `count` fields from the store as indicated below:

```
listeners: {
  show: function(interaction, item, panel){
    var storeItem = item.storeItem;
    panel.update(['<ul><li><b>Offense: </b>' +
      storeItem.get('label') + '</li>', '<li><b>Count: </b> ' +
      storeItem.get('count') + '</li></ul>'].join(''));
  }
}
```

### Update the Chart Panel Title

16. Where indicated by the comment, insert code that update's the chart panel's title with the address that the user typed in the Map panel. Your code should be similar to the following:

```
Ext.getCmp('chartPanel1').getDockedItems()[0].getComponent(0).setTitle(address);
```

17. Save the file and test

– End of Walkthrough --

## Unit Summary



- Ext.Video is a wrapper for an HTML 5 <video> tag
- Ext.Audio is a wrapper for an HTML 5 <audio> tag
- Ext.Map is a wrapper for the Google Maps v3 API
- You must include the Google Maps Javascript to enable Ext.Map functionality
- The Ext.Map().map property enables you to access your Google map directly to execute map API methods
- Sencha Touch Charts enable you to create interactive, vector-based graphs on an HTML5 canvas
- You can use Sencha Charts to create eight different series views:
  - Area
  - Bar
  - Column
  - Scatter
  - Line
  - Pie
  - Radar
  - Gauge

## Unit Review



1. What are the recommended video formats for Ext.Video?
2. What are the recommended audio formats for Ext.Audio?
3. Why is it necessary to Geocode an address?
4. You cannot plot the user's current position on a map (true/false)
5. You cannot customize the map marker (true/false)
6. Which types of charts are supported by Sencha Touch Charts?
7. Which files must you load into your browser in order to begin a Sencha Touch Charts project?
8. You cannot use Array-based data in a Touch Chart. (true/false)

## Lab 6: Adding Video and Maps



During this lab you will develop the campus locator depicted below. You will also add additional functionality to the course browser that displays a video introduction for the class if one is available..

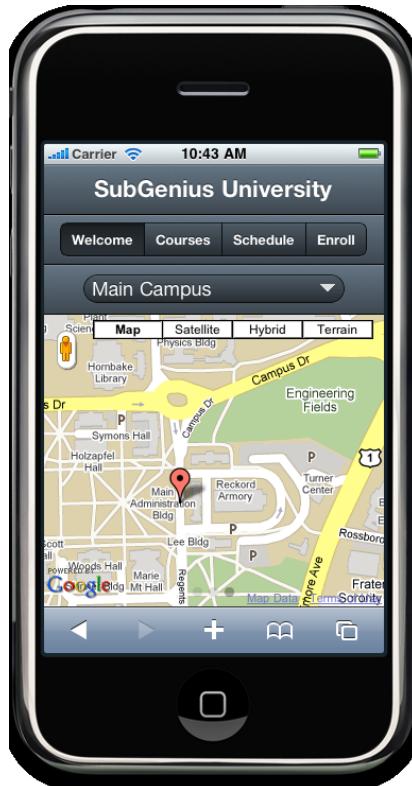


Illustration 12: Campus Location Finder



Illustration 13: Video preview of the Adv ColdFusion class

## Objectives

After completing this lab, you should be able to:

- Add Google Maps functionality to your applications
- Populate a select control with data downloaded from an application server.
- Dynamically instantiate a video panel from within a pre-existing parent panel

## Steps

1. Open **lab/lab6/index.htm** in your **f1st** project. Note that this is the solution from the prior lab.

### Instantiate a Google Map

2. Where indicated by the comment, insert a <script> that loads the Google Map API.
3. Where indicated by the comment, delete the **contentEl** attribute from the locations panel definition
4. Inside the locations panel, add an items array. Inside the items array, instantiate a Google map using the following configuration attributes:
  - id: 'googleMap'
  - xtype: 'map'
  - useCurrentLocation: false
5. Configure the mapOptions for your Google map using the following attributes:
  - center: new google.maps.LatLng (38.909085k -77.036777)
  - zoom: 16
  - mapTypeId: google.maps.MapTypeId.TERRAIN
  - navigationControl: true
  - navigationControlOptions: {
    - style: google.maps.NavigationControlStyle.DEFAULT
}
6. Save the file and test. You should see a Google map appear in the third panel of the **Welcome** section.

### Define a Toolbar to Select a Location

7. Return to your editor
8. Modify the panel that contains your Google map so that it has a toolbar docked at the top.
9. Inside the toolbar that you defined in the previous step, instantiate a SELECT box with an id of 'campusSelector'
10. Center the SELECT box that you defined in the previous step.
11. Save the file and test.

12. Review your toolbar code to ensure it resembles the following;

```
dockedItems: [
  {
    xtype: 'toolbar',
    dock: 'top',
    layout: {pack: 'center'},
    items: [
      {
        xtype: 'selectfield',
        id: 'campusSelector'
      }
    ]
}
```

#### Dynamically Populate a Select Control

13. Review the **plotPoint** function that has been provided for you.
14. In your editor, open up the following data file and review its contents:

`/ftst/json/campuslocations.json`

15. Where indicated by the comment, define a new AJAX request that downloads data from the following url:

`.../.../json/campuslocations.json`

16. Your code should resemble the following:

```
Ext.Ajax.request({
  url: '.../.../json/campuslocations.json',
  success: function(response) {
  },
  failure: function() {
  }
});
```

17. Inside your Ajax success handler, use the `Ext.decode()` function to process the results of the Ajax request and place its value into the variable **aCampusLocations**
18. Underneath the code that you inserted in the prior step, declare a local variable named **aOptions** as an empty Array.

19. Underneath the code that you inserted in the prior step, loop through the **aCampusLocations** array.
- For each entry in the **aCampusLocations** array, define an entry in the **aOptions** array
  - Each entry in the **aOptions** array should be a JavaScript Object containing two properties named **text** and **value**
    - Set the **text** property of the JavaScript object equal to the **TITLE** value of the record retrieved from the Ajax request.
    - Set the **value** property of the JavaScript object equal to the current array index.
20. Underneath the loop that you defined in the prior step, populate the **campusSelector** select box with the **aOptions** array. Your code should resemble the following:

```
Ext.getCmp('campusSelector').setOptions(aOptions, false);
```

21. Set the default value campusSelector to 0 as indicated here:

```
Ext.getCmp('campusSelector').setValue(0);
```

22. Directly underneath the code that you inserted in the prior step, invoke the **plotPoint()** function, passing in values from the first entry in the **aCampusLocations** array.
23. Review your success handler to ensure it resembles the following:

```
success: function(response) {
    aCampusLocations = Ext.decode(response.responseText);
    var aOptions = [];
    for (var i=0; i<aCampusLocations.length; i++) {
        aOptions[aOptions.length] = {
            text: aCampusLocations[i].TITLE,
            value: aOptions.length
        }
    }

Ext.getCmp('campusSelector').setOptions(aOptions, false);
Ext.getCmp('campusSelector').setValue(0);

plotPoint( aCampusLocations[0].LAT,
            aCampusLocations[0].LNG,
            aCampusLocations[0].TITLE,
            aCampusLocations[0].DESCRIPTION,
            Ext.getCmp('googleMap')
        );
}
```

24. Inside the Ajax request failure handler, output an alert box that states the list of locations could not be retrieved. Your failure handler should resemble the following:

```
failure: function() {
    Ext.Msg.alert('Server Error', 'Could not retrieve list of
campus locations');
}
```

25. Save the file and test.

#### Dynamically Instantiate a Video Panel

26. Where indicated by the comment, wrap the **detailCard.update** directive in an IF..ELSE block.
27. If the **VIDEOINTROURL** field from the dataset is empty, execute the **detailCard.update()** method to change the html contents of the panel.
28. If the **VIDEOINTROURL** field from the dataset is NOT empty, dynamically add a video panel to the **detailCard**. Use the following configuration attributes for the video panel:
  - xtype: 'video'
  - url: '../video/' + dataObj.VIDEOINTROURL
  - posterUrl: '../video/' + dataObj.VIDEOINTROPOSTER
29. Invoke **detailCard.doLayout()** to display the newly instantiated video panel
30. Review your code to ensure it resembles the following:

```
if (dataObj.VIDEOINTROURL == '') {  
    detailCard.update("<p id='coursedetailinfo'>" +  
        dataObj.COURSETEASER + "</p>");  
} else {  
    detailCard.add({  
        xtype: 'video',  
        url: '../video/' + dataObj.VIDEOINTROURL,  
        posterUrl: '../video/' + dataObj.VIDEOINTROPOSTER  
    });  
  
    detailCard.doLayout();  
}
```

31. Save the file and test by browsing to the course "Adv ColdFusion"

---

---

# **Unit 7:**

## **Theming your Applications**

### **Unit Objectives**

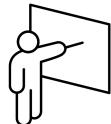
After completing this unit, you should be able to:

- Install SASS and Compass
- Define and deploy efficient, easily maintained CSS files
- Theme your application

### **Unit Topics**

- Using Compass and SASS
- Theming your Sencha Touch Application

## Using Compass and SASS



Sencha recommends that you use SASS and Compass to theme your application.

SASS— Syntactically Awesome Style Sheets, is a style sheet compiler that interprets the SCSS (Sassy CSS) meta language and outputs production-ready CSS.

SASS extends CSS3 by adding nested rules, variables, mixins, selector inheritance, and other useful features. It's available under an MIT license from <http://sass-lang.com>

Compass is a stylesheet authoring tool that uses the SASS language to make your stylesheets smaller and easier to maintain – essentially providing a framework for SASS. Compass Mixins (covered later in this unit) help to smooth out inconsistencies in CSS3 support as indicated by the following example:

SCSS	Generated CSS
<pre>.foo {   @include border-radius(5px); }</pre>	<pre>.foo{   -webkit-border-radius:5px;   border-radius:5px }</pre>

## Installing SASS and Compass

SASS and Compass run on top of Ruby.

- OSX users have Ruby pre-installed.
- Windows users need to download and install Ruby from <http://rubyinstaller.org/>
- Linux users can install Ruby via their package manager

Note that in order to use SASS, you must also have RUBY in your path. Once SASS is installed you can use a command-line utility to automatically translate your SCSS source files into CSS production files whenever the SCSS files are modified.

## Adding Ruby to your Path on OSX

If using OSX, you can add the Ruby path by typing the following at a command prompt:

```
sudo vi /etc/paths
```

You will then add the path to the file using the vi editor. Typically the path resembles the following:

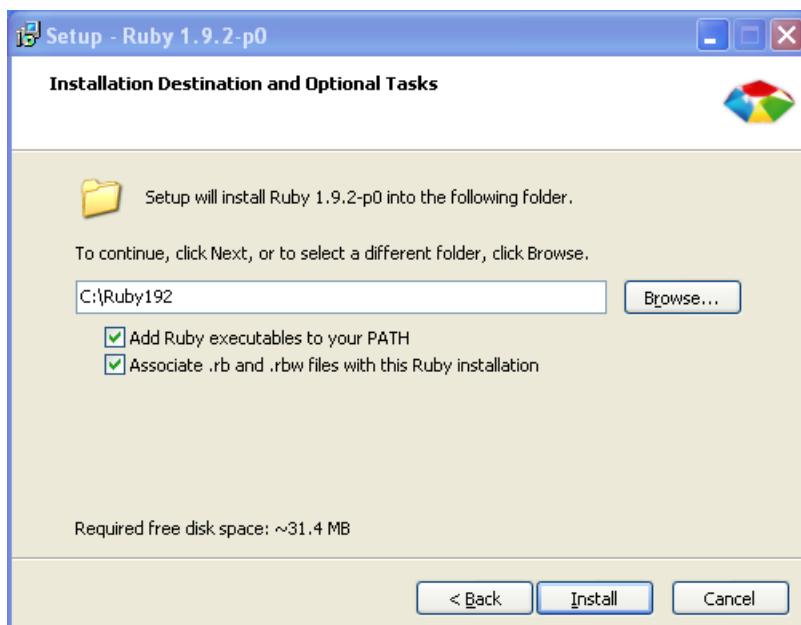
```
/Users/username/.gem/ruby/1.8/bin
```

Note the following commands for the VI editor:

Cmd	Description
*i	Insert text before cursor, until <esc> is pressed
:wq<Enter>	Save the modified file and quit vi
*u	Undo

## Installing Ruby on Windows

If using Windows, turn on the checkbox labeled “Add Ruby executables to your PATH” as indicated below:



**Illustration 1: Ruby Installation Dialog Box**

*Note: You will need to restart Windows after completing the Ruby install process.*

## Deploying SASS and Compass

You can deploy Compass (which includes SASS) by using the gem installer by typing the following at a command prompt:

```
gem install compass
```

## Defining a new Compass project

By default, compass projects automatically include the following assets:

- A default stylesheet for desktop browser screen output
- A default stylesheet for print output
- A configuration file named config.rb

These files are generated automatically by issuing the following command:

```
compass create projectname
```

Where *projectname* is the name of your compass project.

Compass also supports CSS frameworks, including Blueprint ([www.blueprintcss.org](http://www.blueprintcss.org)). Compass will automatically deploy the Blueprint CSS framework if you enter the following command:

```
compass create projectname --using blueprint/basic
```

## Loading Sencha Touch Libraries

In order to use Compass and SASS with Sencha Touch you must modify the config.rb file to automatically include the Sencha Touch libraries.

Typically, you will need to add the following directives to your config.rb file:

```
// get path to .rb file
sass_path = File.dirname(__FILE__)

// specify path for generated css files
css_path = File.join(sass_path,'css')

// load sencha touch extensions by default
load File.join(sass_path,'..','..','..','senchatouch','resources','themes')

// configure output options
output_style = :compressed
environment = :production
```

## Importing Sencha Touch Mixins to your Style Sheet

In order to facilitate theming, you will need to define the following Sencha Touch Mix-ins in your SCSS file. Note that Mix-ins are covered later in this unit. A typical SCSS file used for theming resembles the following:

```
/* modify global variables here */

/* import sencha touch extensions */
@import 'sencha-touch/default/all';
@include sencha-panel;
@include sencha-buttons;
@include sencha-sheet;
@include sencha-picker;
@include sencha-toolbar-forms;
@include sencha-tabs;
@include sencha-toolbar;
@include sencha-carousel;
@include sencha-indexbar;
@include sencha-list;
@include sencha-layout;
@include sencha-form;
@include sencha-loading-spinner;

/* define styles here */
```

Note the following:

The `@import` directive dynamically imports SCSS directives that are contained within an external file. Making additional http requests in a mobile application can be costly, so you can use this directive to compile multiple Compass/SASS files into a single production CSS file.

The `@include` directive dynamically includes a Mixin – a reusable block of style directives that may include parameters.

## Watching for Changes

Use the following command-line syntax to watch a directory for changes, automatically converting any SCSS files over to CSS and placing them into the destination folder specified in the config.rb file.

```
compass watch [dirname]

Steve-Druckers-MacBook-Air:walk stevedrucker$ compass watch theme
>>> Compass is watching for changes. Press Ctrl-C to Stop.
>>> Change detected to: /Applications/ColdFusion9/wwwroot/ftst/walk/theme/src/my
app.scss
identical theme/..../css/src/ie.css
overwrite theme/..../css/src/myapp.css
identical theme/..../css/src/print.css
```

**Illustration 2: Command line utility watches directories for changes**

## Working with SASS

SASS enables you to add programmatic features to your CSS declarations including the following:

- Variables
- Nesting
- Selector Inheritance
- Math functions
- Color Transformations
- Mixins

### Using Variables with SASS

Using SASS you can declare variables that can be used throughout your stylesheet, thereby enabling you to make global changes to your styles by tweaking a single value. Variables begin with a dollar sign (\$) and are declared just like properties. They can have any value that's allowed for a CSS property, such as colors, numbers (with units), or text. The following example illustrates the use of variables and their affect on the generated CSS file:

SASS	Generated CSS
<pre>\$blue: #3bbfce; \$marginleft: 5px;  #myObj {     background-color: \$blue;     margin-left: \$marginleft; }</pre>	<pre>#myObj {     background-color: #3bbfce;     margin-left: 5px; }</pre>

### Using Arithmetic Expressions

SASS supports standard arithmetic expressions (+,-,/%) and will automatically convert between units, if possible. You can nest expressions within parentheses and invoke a number of pre-defined functions.

SASS	Generated CSS
<pre>p { width: 1in + 8pt}</pre>	<pre>p {width: 1.111 in}</pre>
<pre>p {     font: 10px/8px;     \$width: 1000px;     width: \$width/2;     height: (500px/2);     margin-left: 5px + 8px/2px; }</pre>	<pre>p {     font: 10px/8px;     width: 500px;     height: 250px;     margin-left: 9px; }</pre>

## Nesting with SASS

Using nesting you can virtually eliminate the need to code repetitive selectors as indicated by the following example:

SASS	Generated CSS
<pre>body {   font: {     family: comic sans ms;     weight: bold;     size: 1em;   } }  #myObject {   background-color: silver;   .emphasize {     text-decoration: underline;   } }</pre>	<pre>body {   font-family: comic sans ms;   font-weight: bold;   font-size: 1em; }  #myObject {   background-color: silver; }  #myObject .emphasize {   text-decoration: underline; }</pre>

## Defining Selector Inheritance

Using SASS syntax you can easily tell a selector to inherit all of the properties of another selector as described by the following example:

SASS	Generated CSS
<pre>.alert {   font-size: 0.9em;   background-color: yellow; }</pre>	<pre>.alert, .redalert {   font-size: 0.9em;   background-color: yellow; }</pre>
<pre>.alert.important {   font-weight: bold; }</pre>	<pre>.alert.important, .important.redalert {   font-weight: bold; }</pre>
<pre>.redalert {   @extend .alert;   background-color: red; }</pre>	<pre>.redalert {   background-color: red; }</pre>

## Defining and Using Mixins

Mixins allow you to define and reuse blocks of CSS properties. Note that you can also declare Mixins to accept parameterized variables as indicated by the following examples:

SASS	Generated CSS
<pre>@mixin large-text {     font: {         family: Arial;         size: 20px;         weight: bold;     }     color: #ff0000; }  .page-title {     @include large-text;     padding: 4px;     margin-top: 10px; }</pre>	<pre>.page-title {     font-family: Arial;     font-size: 20px;     font-weight: bold;     color: #ff0000;     padding: 4px;     margin-top: 10px; }</pre>
<pre>@mixin sexy-border(\$color, \$width) {     border: {         color: \$color;         width: \$width;         style: dashed;     } }  p {     @include sexy-border(blue, 1in); }</pre>	<pre>p {     border-color: blue;     border-width: 1in;     border-style: dashed; }</pre>

# Walkthrough 7-1: Creating a Compass Project



In this walkthrough, you will perform the following tasks:

- Install Compass
- Initialize a Compass project
- Link a SASS-generated stylesheet to your application
- Change the base color of your application

Note: This exercise assumes that you have Ruby pre-installed on your workstation.



**Illustration 3: Change the color scheme of your application during this walkthrough**

## Steps

### Install Compass

1. If you are on a Windows workstation, download and install Ruby at the following URL:

<http://rubyinstaller.org>

2. Open a command prompt and type the following:

```
gem install compass
```

3. After the installation process has completed, type the following code to verify your installation:

```
compass version
```

### Create a Compass Project

4. In your command prompt, change directory to your project's WALK web root.

```
cd /apache/htdocs/ftst/walk
```

5. Create a Compass project by typing the following command:

```
compass create theme
```

6. Return to your editor

7. Right-click on the walk directory in your project and select Refresh

### Configure Compass for use with Sencha Touch

8. Right-click on the `/ftst/walk/theme/config.rb` file and select **Open With...Text Editor**
9. At the top of the file, add the following directives to get the path to the `config.rb` file:

```
sass_path = File.dirname(__FILE__)
```

10. Underneath the code that you inserted from the prior step, insert a Ruby directive to load the Sencha Touch framework automatically:

```
load File.join(sass_path, '..', '..', '..', 'senchatouch', 'resources', 'themes')
```

11. Configure the destination directory for your generated CSS files by adding the following directive underneath the code that you inserted in step 9:

```
css_path = File.join(sass_path, 'css')
```

12. Save the file
13. Return to your command prompt
14. Have Compass watch your theme folder for changes by typing the following command:

```
compass watch theme
```

### Use Compass to Generate a Style Sheet

15. Copy the file `/ftst/snippets/myapp.scss` to `/ftst/walk/theme/sass`
16. Open `/ftst/walk/theme/sass/myapp.scss` in your editor and review its contents with your instructor

### Demonstrate Theming

17. In the `myapp.scss` file, where indicated by the comment, insert the following command to change the base color of your application:

```
$base-color: #7F3115;
```

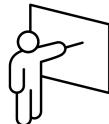
18. Save the file
19. Review the output from your command window to see if an updated CSS file was generated
20. Using your editor, open `/walk7-1/index.html`
21. Where indicated by the comment, link to the generated CSS file using the following syntax:

```
<link rel="stylesheet"
      href="../../theme/css/sass/myapp.css"
      type="text/css" />
```

22. Save the file and browse

– End of Walkthrough --

# Theming your Sencha Touch Application



Most mobile applications can be made to conform to branding standards by using the following techniques:

- Modifying Sencha Touch SASS Variables
- Invoking Sencha Touch Mixins
- Theming Buttons and Toolbars

## Working with Sencha Touch SASS Variables

The following variables may be tweaked in your SCSS file to modify the look and feel of your application:

- \$active-color
- \$alert-color
- \$base-color
- \$base-gradient
- \$bright-color
- \$button-height
- \$button-radius
- \$button-stroke-weight
- \$button-gradient
- \$carousel-indicator-size
- \$carousel-indicator-spacing
- \$carousel-track-size
- \$form-bg-color
- \$form-field-bg-color
- \$form-light
- \$form-dark
- \$form-label-width
- \$form-field-height
- \$form-spacing
- \$form-textarea-height
- \$form-thumb-size
- \$form-toggle-size
- \$form-thumb-space
- \$form-fieldset-radius
- \$form-slider-size
- \$global-row-height
- \$include-button-uis
- \$include-default-icons
- \$include-default-uis
- \$include-form-sliders
- \$include-html-style
- \$include-highlights
- \$include-tabbar-uis
- \$include-toolbar-uis
- \$include-top-tabs
- \$include-bottom-tabs
- \$light-tab-color
- \$light-tab-active
- \$page-bg-color
- \$sheet-bg-color
- \$sheet-button-spacing
- \$sheet-padding
- \$tabs-bottom-radius
- \$tabs-bottom-icon-size
- \$tabs-bottom-active-gradient
- \$tabs-bottom-gradient
- \$tabs-dark-color
- \$tabs-dark-active-color
- \$top-tab-height
- \$toolbar-spacing
- \$toolbar-input-bg
- \$toolbar-input-color
- \$toolbar-input-height
- \$toolbar-gradient
- \$toolbar-highlights
- \$toolbar-icon-size

## Changing the Base Color

As depicted in the screen shots below, the \$base-color variable controls the overall color palette of your application. Changing this variable is the fastest way to apply your brand colors.

Changing the Base Color	
Default	\$base-color: #7F3114;

## Changing the Base Gradient

The \$base-gradient variable influences CSS3 gradient stops. Valid options include bevel, glossy, recessed, and matte (default).

Changing the Base Gradient	
Default (matte)	\$base-gradient: glossy;

## Changing the Alert and Confirm colors

The following variables may be set to override default button colors:

Variable	Description
\$alert-color	Configures the button color for items with an assigned ui of 'decline'
\$confirm-color	Configures the button color for items with an assigned ui of 'confirm'

The following example illustrates the effect of setting these variables:

### MyApp.SCSS:

```
$base-color: green;
$alert-color: purple;
$confirm-color: yellow;
```

### MyApp.JS:

```
{
    xtype: 'toolbar',
    items: [
        { ui: 'decline', text: 'Decline' },
        { ui: 'decline-round', text: 'Decline' },
        { ui: 'decline-small', text: 'Decline' },
        { ui: 'round', text: 'Decline', badgeText: 2 },
        { ui: 'confirm', text: 'Confirm' },
        { ui: 'confirm-round', text: 'Confirm' },
        { ui: 'confirm-small', text: 'Confirm' }
    ]
}
```

### Screen Shot:



Note the following:

- The \$alert-color and \$confirm-color also affect the buttons that appear in the pickers that are invoked by the selectfield and datefield form elements
- The \$alert-color is used to style the badge text for a button
- In a future release, \$alert-color will be applied to form elements that fail data validation.

## Selectively Disabling Your Theming Settings

Occasionally you may want to temporarily disable your custom theming during your debugging process. You can disable your custom theming by setting the following variable:

```
$include-highlights: false;
```

In addition, you can selectively disable theming for specific ui constructs by assigning false to the following variables:

Variable	Description
\$toolbar-highlights	Disable theming for toolbars
\$include-tabbar-highlights	Disable theming for tab bars

## Defining Custom Style Classes with Mixins

You can use the following Mixins to help you define a custom classes:

- **@include background-gradient(\$bg-color, \$type)**  
Defines a background gradient.  
Valid types include matte, bevel, glossy, recessed, and flat.
- **@include color-by-background (\$bg-color, \$contrast: 100%)**  
Sets a foreground color, adjusts the specified color based on the contrast percentage that is specified.
- **@include bevel-by-background(\$bg-color)**  
Defines a text-shadow that simulates a beveled effect
- **@include mask-by-background(\$bg-color, \$contrast, \$style)**  
Defines a background color and gradients. Valid \$style values include matte, bevel, glossy, recessed, and flat.

*Note: The Mixins are defined in the following Sencha Touch directory:  
resources/themes/stylesheets/sencha-touch/default/\_mixins.scss*

## Using the background-gradient Mixin

The following SCSS definition:

```
.myClass { @include background-gradient(#5291c5,glossy); }
```

Generates the following CSS output:

```
.myclass{  
background-color:#5291c5;  
background-image:  
-webkit-gradient(linear, 0% 0%, 0% 100%,  
color-stop(0%, #8bb5d8),  
color-stop(50%, #659dc5),  
color-stop(51%, #5291c5),  
color-stop(100%, #4085be)  
);  
background-image:  
linear-gradient(top,  
#8bb5d8 0%,  
#659dc5 50%,  
#5291c5 51%,  
#4085be 100%)  
}
```

## Using the color-by-background Mixin

The color-by-background Mixin automatically chooses the appropriate foreground color that is automatically based on the amount of contrast that is desired from the background color.

Therefore, the following SCSS definition:

```
.myClass {@include color-by-background(#5291c5,10%); }
```

Results in the following CSS declaration:

```
.myClass { color: #3977ab; }
```

## Using the bevel-by-background Mixin

This Mixin is a proxy for defining a text-shadow.

The following SCSS definition:

```
.myClass {  
@include bevel-by-background(yellow);  
}
```

Results in the following CSS declaration:

```
.myClass {  
text-shadow: rgba(0, 0, 0, 0.5) 0 -0.08em 0;  
}
```

## Using the mask-by-background Mixin

The following SCSS definition:

```
.myClass {
  @include mask-by-background(#5291C5, 10%, glossy);
}
```

Results in the following CSS declaration:

```
.myClass {
  background-color: #3977ab;
  background-image:
    -webkit-gradient(
      linear, 0% 0%, 0% 100%,
      color-stop(0%, #659dcb),
      color-stop(50%, #4085be),
      color-stop(51%, #3977ab),
      color-stop(100%, #336a98)
    );
  background-image:
    linear-gradient(
      top,
      #659dcb 0%,
      #4085be 50%,
      #3977ab 51%,
      #336a98 100%
    );
}
```

## Applying Custom Style Classes to Sencha Touch Components

Every Sencha Touch component supports at least one configuration option that enables you to specify a custom style class that overrides its default ui properties.

Most objects support the same style-related properties as the Component class, which are listed below:

Component Theming-Related Properties	
Property	Description
cls	An optional extra CSS class that will be added to this component's Element (defaults to ""). This can be useful for adding customized styles to the component or any of its children using standard CSS rules.
componentCls	CSS Class to be added to a components root level element to give distinction to it via styling.
floatingCls	The class that is being added to this component when its floating. (defaults to x-floating)

Component Theming-Related Properties	
style	A custom style specification to be applied to this component's Element.
styleHtmlCls	The class that is added to the content target when you set styleHtmlContent to true. Defaults to 'x-html'

Applying a background gradient to a panel becomes a rather straightforward affair.

Given the following definition in your scss file:

```
$myBgColor: #5291C5;
.myClass {
    @include background-gradient($myBgColor, glossy);
    @include bevel-by-background($myBgColor);
}
```

You could use the class in a Panel as follows:

```
var mainPnl = new Ext.Panel({
    cls: 'myClass',
    html: 'Some Text'
})
```

## Theming Buttons and Toolbars

You can theme buttons and toolbars by invoking a series of Sencha Touch SASS Mixins.

- Use the sencha-button-ui mixin to define a custom ui for a button
- Use the pictos-iconmask mixin to define an icon class for use in a button
- Use the sencha-toolbar-ui mixin to define a custom ui for a toolbar

### Theming Buttons with `sencha-button-ui`

Invoke the sencha-button-ui mixin in you SCSS file to define a custom style. This mixin accepts the following parameters, listed in order:

- The ID of the new ui class to be generated
- The background color of the button
- The visual effect (matte, bevel, glossy, recessed, and flat)

The following example illustrates defining a custom button theme:

**In your SCSS:**

```
@include sencha-button-ui('silver', #c0c0c0, 'glossy');
```

**In your JavaScript:**

```
{
    xtype: 'toolbar',
    items: [
        { ui: 'silver', text: 'Hey Now'},
        { ui: 'silver-round', text: 'I\'m round'},
        { ui: 'silver-small', text: 'I\'m small'}
    ]
}
```

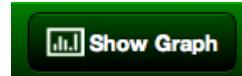
**Result:**

Note the following:

- The Mixin automatically generates a “round” and “small” sub-style
- The foreground color is chosen automatically and is based on the color of the background that you specified.

## Defining Buttons with Custom Icons

As previously mentioned in this course, Sencha Touch ships with over 300 icons. While many of these icons are deployable through the iconCls property of the Ext.Button component as demonstrated in unit 3, others may only be invoked only by using the pictos-iconmask mixin.



Sencha Touch icons are located in the following directory:

```
resources/themes/images/default/pictos
```

The following SASS code defines a new iconCls named chart1 that makes the chart1.png file located in the pictos repository available for use in a button:

```
@include pictos-iconmask('chart1');
```

You can subsequently refer to the image in your button definition as indicated below:

```
xtype: 'toolbar',
items: [
    {text: 'Show Graph', iconCls: 'chart1', iconMask true}
]
```

## Theming Toolbars

Use the sencha-toolbar-ui mixin to define a custom ui for a toolbar. This mixin supports the following arguments, listed in order:

- UI Identifier
- Background Color
- Type (supported types include matte, bevel, glossy, recessed, flat)

The following example illustrates defining and referencing a custom ui for a toolbar:

### In your SCSS File:

```
@include sencha-toolbar-ui('red', #ff0000, 'bevel');
```

### In your JavaScript File:

```
{
  xtype: 'toolbar',
  ui: 'red',
  items: [
    {
      text: 'Show Graph',
      iconCls: 'chart1',
      iconMask: true
    }
    /* other button defs ... */
  ]
}
```

### Results:



## Theming Tab Bars

Use the sencha-tabbar-ui mixin to define a custom ui for a tab bar. This mixin supports the following arguments, listed in order:

- UI Identifier
- Background Color
- Type (supported types include matte, bevel, glossy, recessed, flat)
- Active Tab Color

The following example illustrates defining and referencing a custom ui for a toolbar:

### In your SCSS File:

```
@include sencha-tabbar-ui('secondarytabbar',
                           #003366,
                           'bevel',
                           #ffffff);
```

### In your JavaScript File:

```
var feedsPnl = new Ext.TabPanel(
{
    fullscreen: true,
    ui: 'dark',
    sortable: true,
    tabBar: {
        dock: 'bottom',
        ui: 'secondarytabbar',
        layout: {pack: 'center'}
    },
    ...
})
```

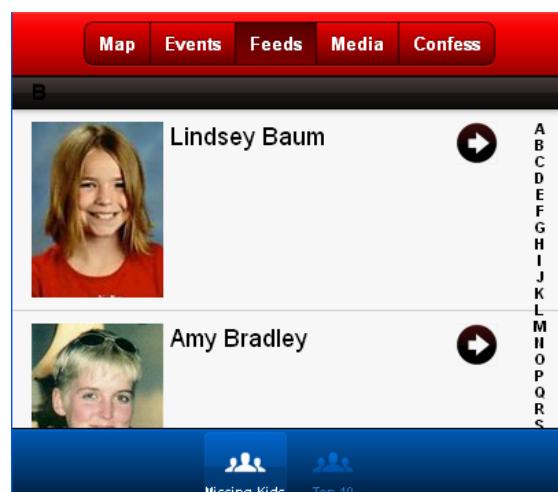


Illustration 4: Theming a tab bar

## Theming Lists

To theme a list, you should modify the following variables described below:

SASS Variable	Description
\$list-pressed-color	The color of a list item when it is clicked and the mouse button is in a “down” state
\$list-active-color	The color of the currently selected list item

## Optimizing Your CSS

The CSS file that is generated by Compass for Sencha Touch can be quite large – up to 200K or more. Due to the bandwidth limitations of mobile devices, it falls upon you to reduce this file size as much as possible.

Typically you will be able to shrink the file significantly by performing the following actions:

- Remove unused components
- Remove Unused Images
- Remove Unused UI definitions
- Set the output style to compressed

### Removing Unused Components

You should remove references in your SCSS file to components that are not used within your application. For example, if your application does not contain any data entry forms, removing the sencha-form and sencha-toolbar-form references will reduce your CSS by approximately 30K.

Compiling your own Compass/SASS file using this technique also enables you to remove the <link> tag in your .html file that references the sencha-touch.css file.

```

11
12@import 'compass';
13@import 'sencha-touch/default/all';
14@include sencha-panel;
15@include sencha-buttons;
16@include sencha-sheet;
17@include sencha-picker;
18@include sencha-toolbar-forms;
19@include sencha-tabs;
20@include sencha-toolbar;
21@include sencha-carousel;
22@include sencha-indexbar;
23@include sencha-list;
24@include sencha-layout;
25@include sencha-form;
26@include sencha-loading-spinner;
27

```

**Illustration 5: Delete references to components that you are not using in your application**

## Removing Unused Images and UI Definitions

Sencha Touch optimizes the performance of your application by base-64 encoding frequently used images into your CSS file. This technique helps minimize the number of http requests that your mobile browser must dispatch and therefore speeds up performance at runtime.

Base-64 encoded images are represented in CSS using the following syntax:

```
<style type="text/css">
  div.mycontainer {
    background-image:
      url(data:image/gif;base64,R0lGODdhAQAg...);
  }
</style>
```

The problem with using this approach is that the base-64 encoding process typically increases the image file size by approximately 33%. You can suppress the inclusion of the default images by including the following directives in your SCSS file which will shrink your generated CSS file by approximately 40%:

```
$include-default-icons: false;
$include-button-uis: false;
```

Once you have suppressed the default icons and ui's you must remember to manually add references to any icons and ui's that your application requires by using the following mixins:

```
@include pictos-iconmask('imagename');
@include sencha-button-ui('uiname', color);
```

*Note: The default icon and button ui references are contained in the following Sencha Touch file:*

*/resources/themes/stylesheets/sencha-touch/default/widgets/\_buttons.scss*

## Compressing your generated CSS file

Compass will automatically compress your generated CSS file and remove any comments by including the following commands at the bottom of your config.rb file:

```
output_style = :compressed
environment = :production
```

By using these processing directives and applying all of the other optimizations listed in this section, you should be able to compress your production CSS file to approximately 75k.

## Walkthrough 7-2: Theming the CrimeFinder



In this walkthrough you will perform the following tasks:

- Remove the CrimeFinder toolbar
- Brand the application using an approved color scheme
- Deploy a new button icon
- Optimize the production CSS



Illustration 6: The approved color palette for the CrimeFinder application

## Steps

### Define and Apply a Base Color Scheme

1. Open `/walk/theme/sass/myapp.scss` in your code editor
2. Set the base color of the application to `#999999`

```
$base-color: #999999;
```

3. Save the file and browse the application
4. Return to `myapp.scss` in your code editor
5. Add a variable declaration to set the base gradient to glossy

```
$base-gradient: glossy;
```

6. Save the file and browse your application
7. Return to **myapp.scss** in your code editor

#### Define and Deploy Toolbar Themes

8. At the bottom of the file, define a new toolbar ui named **primarytoolbar** that uses the color **#cc0000** with a **glossy** effect
9. At the bottom of the file, define a new toolbar ui named **secondarytoolbar** that uses the color **#003366** with a **beveled** effect
10. At the bottom of the file, define a new toolbar ui named **tertiarytoolbar** that uses the color **#FFFFFF** with a **beveled** effect
11. Review your code to ensure it appears as follows:

```
@include sencha-toolbar-ui('primarytoolbar',#cc0000,'glossy');  
@include sencha-toolbar-ui('secondarytoolbar',#003366, 'bevel');  
@include sencha-toolbar-ui('tertiarytoolbar',#ffffff, 'bevel');
```

12. Save the file
13. Return to your code editor and open /walk/walk7-2/index.html
14. Where indicated by the comment, add a **ui** configuration property to the main toolbar. Set the value of ui to **primarytoolbar**.
15. Save the file and browse. Your main toolbar should now be colored red.
16. Where indicated by the comment, add a ui configuration property to the address toolbar on the map panel. Set the value of ui to **secondarytoolbar**.
17. Save the file and browse. The address toolbar should now be colored blue.
18. Return to your code editor.
19. Where indicated by the comment, add a **ui** configuration property to the bottom toolbar on the **Events** panel. Set the value of ui to **secondarytoolbar**.
20. Where indicated by the comment, add a **ui** configuration property to the bottom toolbar on the **Confess** panel. Set the value of ui to **secondarytoolbar**.
21. Save the file and browse.
22. Where indicated by the comment, add a **ui** configuration property to the bottom toolbar on the **Feeds** panel. Set the value of ui to **secondarytoolbar**.
23. Remove the toolbar at the top of the application that contains the “CrimeFinder” title
24. Save the file and browse
25. Return to your editor

### Theme the Tab Bar

26. At the bottom of the myapp.scss file, define a new ui for a tabbar. Use the following attributes:

- UI name: 'secondarytabbar'
- Background Color: #003366
- Effect: 'bevel'
- Active Color: #ffffff

27. Verify that your code appears as follows:

```
@include sencha-tabbar-ui('secondarytabbar',#003366, 'bevel', #ffffff);
```

28. Save the file
29. Return to **index.html** in your editor
30. Where indicated by the comment, insert a **ui** attribute with a value of **secondarytabbar**
31. Save the file and browse

### Add custom icons

32. Using your file explorer, browse the contents of  
c:/apache/htdocs/senchatouch/resources/themes/images/default/pictos
33. Return to your editor and open **myapp.scss**
34. At the bottom of the file, use the appropriate mixin to utilize the **locate3.png** icon in your application. Your code should appear similar to the following:

```
@include pictos-iconmask('locate3');
```

35. At the bottom of the file, use the appropriate mixin to utilize the **user\_list.png** icon in your application. Your code should appear similar to the following:

```
@include pictos-iconmask('user_list');
```

36. Save the file
37. Open **/walk/walk7-2/index.html**
38. Where indicated by the comment, change the **iconCls** value to **'locate3'**
39. Where indicated by the comment, change the **iconCls** value to **'user\_list'**
40. Save the file and browse

### Optimize the Generated CSS File

41. In Aptana, right click on **/walk/theme/css/sass/myapp.css** and select **Properties**. Note the size of the file

42. Right-click on **/walk/theme/config.rb** and select **Open With...Text Editor**

43. Add the following lines of code to the bottom of the file in order to generate a compressed css file

```
output_style = :compressed  
environment = :production
```

44. Save the file

45. Open the Command Window that is currently running Compass

46. Press **Ctrl+C** and terminate the batch job

47. Restart Compass by typing `compass watch theme` and press Enter

48. Return to Aptana and open **/walk/theme/sass/myapp.scss**

49. At the top of the file, instruct Compass to not load the icon library by typing the following commands:

```
$include-default-icons: false;
```

50. Save the file

51. In Aptana, right click on **/walk/theme/css/sass/myapp.css** and select **Properties**. Note the size of the file.

52. Browse **/walk/walk7-2/index.htm**. Do you notice any differences in the application?

53. Open **/walk/walk7-2/index.htm** in Aptana

54. Delete `<link>` tag that invokes **sencha-touch-debug.css**

55. Save the file and browse. Do you notice any differences in the application?

56. Return to Aptana and open **/walk/theme/sass/myapp.scss**

57. Add the following entries at the bottom of the file to restore the lost icons:

```
@include pictos-iconmask('locate');  
@include pictos-iconmask('organize');
```

58. Save the file

59. Browse **/walk/walk7-2/index.htm**. Your missing icon buttons should be restored.

– End of Walkthrough --

# Unit Summary



- Sencha Touch uses Compass and SASS to generate CSS files
- SASS and Compass are built on top of Ruby. OSX users have Ruby pre-installed, Windows users will need to download the installer.
- SASS enables you to more easily maintain your production CSS by allowing for variable definitions, conditional logic, math functions, mixins, and more in your SCSS file
- You can quickly apply a custom color scheme to your application by setting the \$base-color and \$base-gradient variables
- There are a number of Sencha Touch mixins available that reduce the complexity of defining cross-browser compatible CSS3 effects
- Use the **pictos-iconmask** mixin in order to deploy any of the two hundred icons that ship with Sencha Touch
- Use the **sencha-button-ui** mixin to define a custom ui theme for your buttons
- Remember to remove unused component, icon, and ui references from your SCSS prior to launch.

## Unit Review



1. You must install Ruby on your production web server in order to use SASS and Compass (true/false)
2. List three major benefits of using SASS
3. What is the syntax for defining a SASS variable?
4. What is the procedure for translating a SCSS file into a CSS file
5. Illustrate the syntax for defining and using a mixin
6. How can you define a custom ui theme?

# Lab 7: Theming SubGenius University



During this lab you will define a custom theme for the SubGenius University application as depicted below.

## Objectives

During this lab you should:

- Remove the SubGenius University toolbar
- Brand the application using an approved color scheme
- Define a custom ui theme and use it in a toolbar
- Deploy a custom icon button

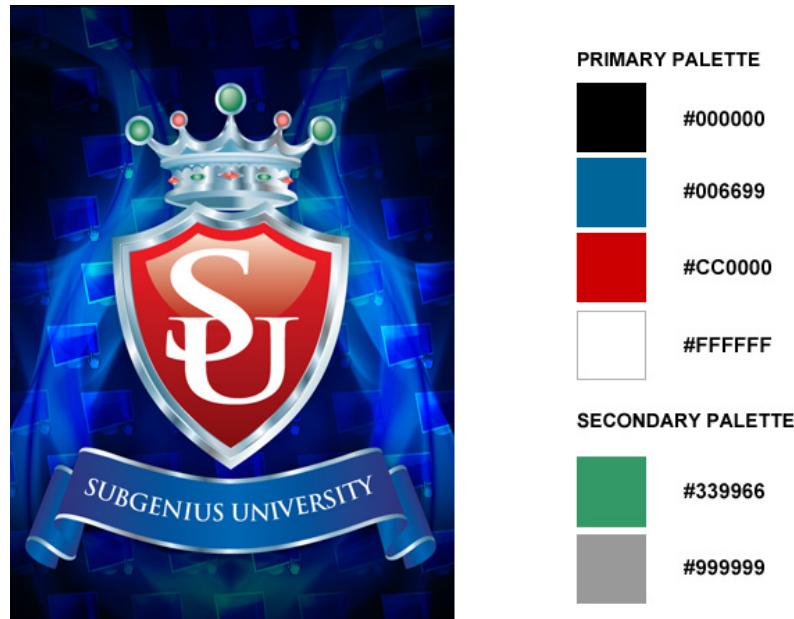


Illustration 7: Approved Color Palette for SubGenius University

## Steps

1. Open a Command prompt and navigate to your lab folder using the following command:

```
cd /apache/htdocs/ftst/lab
```

2. Create a Compass project by typing the following command:

- ```
compass create theme
```
3. Return to Aptana
  4. Right-click on the **walk** directory in your project and select **Refresh**

#### Configure Compass for use with Sencha Touch

5. Right-click on the **/ftst/walk/theme/config.rb** file and select **Open With...Text Editor**
6. At the top of the file, add the following directives to get the path to the config.rb file:

```
sass_path = File.dirname(__FILE__)
```

7. Underneath the code that you inserted from the prior step, insert a Ruby directive to load the Sencha Touch framework automatically:

```
load File.join(sass_path, '..', '..', '..', 'senchatouch', 'resources', 'themes')
```

8. Configure the destination directory for your generated CSS files by adding the following directive underneath the code that you inserted in step 9:

```
css_path = File.join(sass_path, 'css')
```

9. Save the file
10. Return to your command prompt
11. Have Compass watch your theme folder for changes by typing the following command:

```
compass watch theme
```

12. Using the techniques described in this chapter, theme the SubGenius University application. Refer to the color guide on page 7-30 for guidance.

– End of Lab --

---

---

# **Unit 8:**

## **Optimizing your Applications**

### **Unit Objectives**

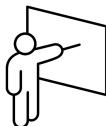
After completing this unit, you should be able to:

- Cache application fields locally on a device
- Cache data locally on a device
- Apply object-oriented paradigms to your development effort
- Use JSBuilder to optimize your codebase for production
- Use PhoneGap to produce a native application

### **Unit Topics**

- Caching Application Files Locally
- Caching Data Locally
- Making your Code Base Manageable with SMVC
- Optimizing your Distribution with JSBuilder
- Compiling Native Apps with PhoneGap

## Caching Application Files Locally

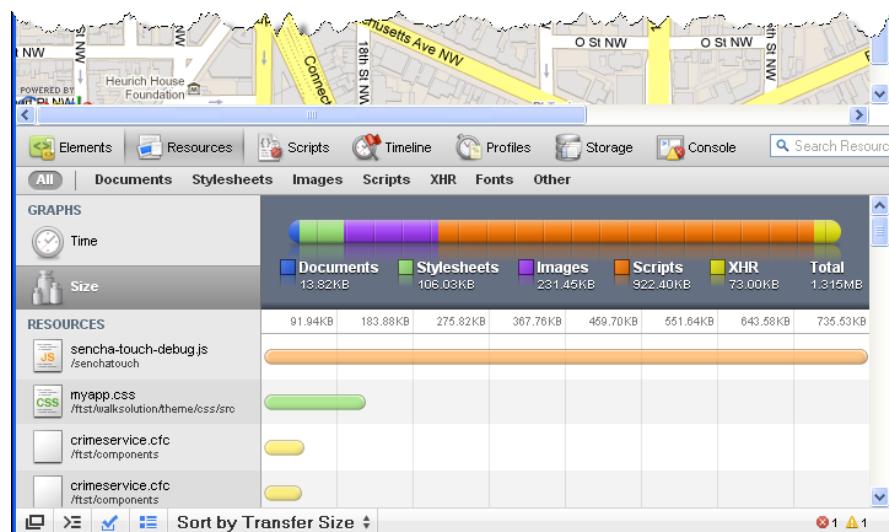


HTML 5 enables you to cache files for off-line access. It also includes a number of JavaScript methods to determine whether or not a network connection is available to the user agent.

### Determining which files should be cached

The Resources panel of Safari and Google Chrome, depicted below, displays a list of all files that were loaded by your application. Typically you will want to cache the following:

- Your application start page (typically index.htm)
  - The Sencha touch javascript library
  - Your application's CSS file
  - Media (images, audio, video) that are referenced by your application
- Content that you should not cache typically includes dynamic data that changes frequently including:
- URL's that return dynamic data sets such as web services
  - Server-side application pages
  - XML/RSS feeds
  - Form action pages



**Illustration 1:** Use the resources panel to identify which files should be cached

## Defining the files to be cached

You can cache an HTML application locally by adding the manifest attribute to the <html> tag as indicated below:

```
<!DOCTYPE HTML>
<html manifest="cache.manifest">
  ...

```

The attribute takes a URI to a manifest, which specifies which files are to be cached. The manifest has a `text/cache-manifest` MIME type containing a list of files to be cached. A typical file might resemble the following:

```
CACHE MANIFEST
#version 1
index.html
  ../theme/css/src/myapp.css
  /senchatouch/sencha-touch-debug.js
  ../../images/cf_background.png
  ../../images/fbi.jpg
  ../../video/poster1.png

NETWORK:
http://maps.gstatic.com/
http://maps.google.com/
http://maps.googleapis.com/
http://mt1.googleapis.com/
http://mt0.googleapis.com/
http://gg.google.com/
  ../../components/crimeservice.cfc
http://www.fbi.gov/

FALLBACK:
/ /notavailable.html
```

Note the following:

- The first line of the file MUST be `CACHE MANIFEST`
- The web server must send a `text/cache-manifest` MIME type
- The manifest is divided into three sections.
  - The MANIFEST section details which files should be persisted
  - The NETWORK section details which files should be ignored
  - The FALLBACK section details the file to be substituted if a requested file is not in the app cache and the browser is not online.
- Optional comments must start with a pound sign (#)
- If your application uses Google Maps, you must include the google-related entries in the NETWORK section listed in the prior example.
- In this example, any URL that is not referenced explicitly in the .manifest file will fail to load.

## Loading files not referenced by your manifest file

When connected to the internet, if you want to load files that are not referenced by the manifest file, add a \* to the NETWORK section as described in the following example:

```
CACHE MANIFEST
#version 1
index.html
./theme/css/src/myapp.css
/senchatouch/sencha-touch-debug.js
....../images/cf_background.png
....../images/fbi.jpg
....../video/poster1.png

NETWORK:
*

FALLBACK:
/ /notavailable.html
```

## Configuring your Web Server to Support Application Caching

You will likely need to configure your web server to support the text/cache-manifest mime type. For example, you must add the following entry to your Apache mime.types file:

```
text/cache-manifest manifest

Alternately, in Apache, you can create a .htaccess file in the root of
your site that contains the following directives:

AddType text/cache-manifest .manifest

<Files cache.manifest>
ExpiresActive On
ExpiresDefault "access"
</Files>
```

In this example, the AddType directive instructs Apache to transmit the appropriate mime type for all files that have a .manifest extension. The <Files> directives instruct the web server to transmit an accurate last updated date on the cache.manifest file.

## Troubleshooting Problems with Application Caching

Troubleshooting application caching can be a tricky affair due to the following:

- If any of the files listed on the manifest are not found, no caching will occur
- While waiting for the manifest file, the browser will proactively load the site from cache. Therefore, changes to your cache manifest are only acted upon after the SECOND refresh from when the file was modified.
- You cannot specify an expiration date/time for files in the cache.
- You can expire the cache by making a change to the manifest file. Any change to the file causes all files to be recached.
- Manually clearing the browser cache from the browser menu does NOT force all files to be recached.

## Inspecting the Application Cache

Google Chrome allows you to inspect the files in your application cache by accessing Tools > Developer Tools > Application Cache. Safari

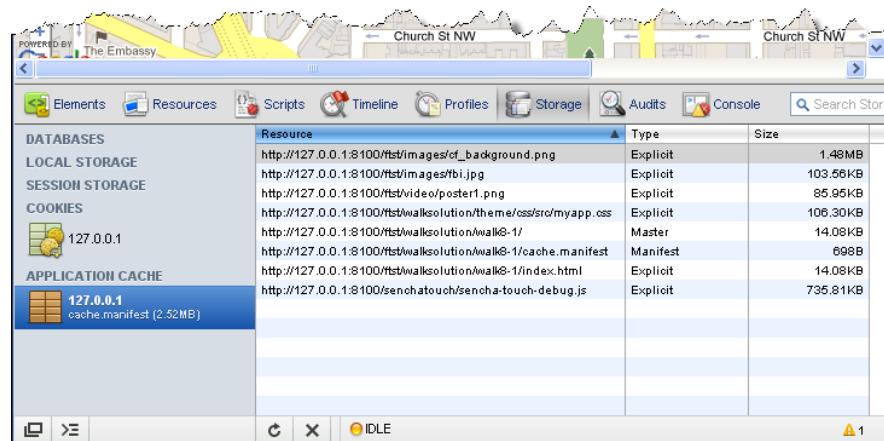


Illustration 2: Inspecting the Application Cache in Google Chrome

## Detecting Network Access using navigator.onLine

Use the `navigator.onLine` property to determine if the browser has network access. You can use this property to alter your business logic to save data locally instead of to the network or to hide services, such as mapping, that are completely dependent on network access.

The following example illustrates using the `navigator.onLine` property to change the options available within your application:

```
if (navigator.onLine) {
    var mainTB = new Ext.Toolbar(
        {
            dock: 'top',
            layout: {pack: 'center'},
            ui: 'primarytoolbar',
            items: [
                {
                    margin: 'auto',
                    xtype: 'segmentedbutton',
                    items: [
                        {text: 'Map', handler: tapHandler,
                         pressed: true},
                        {text: 'Events', handler: tapHandler},
                        {text: 'Feeds', handler: tapHandler},
                        {text: 'Media', handler: tapHandler},
                        {text: 'Confess', handler: tapHandler}
                    ]
                }
            );
} else {
    var mainTB = new Ext.Toolbar(
        {
            dock: 'top',
            layout: {pack: 'center'},
            ui: 'primarytoolbar',
            items: [
                {
                    margin: 'auto',
                    xtype: 'segmentedbutton',
                    items: [
                        {text: 'Media', handler: tapHandler},
                        {text: 'Confess', handler: tapHandler,
                         pressed: true}
                    ]
                }
            ]
        }
    )
}
```

## Determining Network Access via Events

The Window object dispatches online and offline events as indicated by the following example:

```
<script language="javascript">

    function setConnectStatus(e) {
        if (!e) e = window.event;

        if (e.type == 'online' ) {
            alert( 'The browser is ONLINE.' );
        }
        else if (e.type == 'offline' ) {
            alert( 'The browser is OFFLINE.' );
        }
        else {
            alert( 'Unexpected event: ' + e.type );
        }
    }

    window.onload = function() {
        document.body.ononline = setConnectStatus;
        document.body.onoffline = setConnectStatus;
    }

</script>
```

## Walkthrough 8-1: Caching Application Source



In this walkthrough, you will perform the following tasks:

- Cache the application code for offline use
- Verify the application has been cached
- Add code to detect whether the application is connected to the internet

### Steps

#### Configure the Cache Manifest

1. Open /walk/walk8-1/index.html in your editor
2. Change the <!DOCTYPE> declaration to the following:

```
<!DOCTYPE HTML>
```

3. Add a manifest attribute to the <html> tag that references a file named cache.manifest as illustrated below:

```
<html manifest="cache.manifest">
```

4. Save the file
5. Create a new file in the /walk/walk8-1 directory named cache.manifest
6. Copy the contents of /snippets/cache.manifest into **/walk/walk8-1/cache.manifest** and review its contents with your instructor.

#### Configure Apache to Support the MIME type

7. Create a new file in the /walk/walk8-1 directory named **.htaccess**
8. Add the following entry to your **.htaccess** file:

```
AddType text/cache-manifest .manifest
```

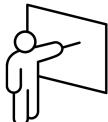
9. Save the file
10. Restart the Apache service

#### Test the Application Cache

11. Open **/walk/walk8-1/index.html** using Google Chrome
12. Reload the page in Google Chrome
13. Select **Tools > Developer Tools** from the **Customize and Control Google Chrome** button (wrench icon)
14. Click on the **Storage** button

15. Click on the **Application Cache** tab
16. Review the contents with your instructor
17. Disable your Apache web server
18. Refresh the application in the browser. Note that many of the application features continue to function despite not having a connection to the server.

## Caching Data Locally



HTML5 enables you to store data locally on the client browser using three separate mechanisms. Each of these enables you to make your application less reliant on a fast connection to your web server.

- HTML5 Local Storage enables you to store keyname/value pairs on the device. Stored data persists on your device, even after a power-down. It is represented in Sencha Touch by the `LocalStorageProxy` class
- HTML5 Session Storage also enables you to store text strings, or abstract data types represented in JSON. However, unlike local storage, data stored using this technique is deleted once the browser is closed. In Sencha Touch, HTML5 session storage is accessible through the `SessionStorageProxy` class.
- You can store relational data in a local SQLite database. There is no abstraction for this through Sencha Touch, however, you can still access the SQLite database through native Javascript API method calls.

## Using Local Storage

You can think of Local Storage as http cookies on mega-steroids. Whereas cookies have an inherent size limitation of 4K per domain, local storage has no such restrictions. Like cookies, data saved into local storage is represented as strings. Syntactically, placing data into local storage is simpler than working with cookies and you can add event listeners to the storage area that will fire whenever content is modified.

### Defining a LocalStorageProxy

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

LocalStorage is extremely useful for saving user-specific information without needing to build server-side infrastructure to support it. Alternately, it can be used to temporarily store information while the browser is waiting for a connection to the server to become available.

Typically you will attach the `localStorage` proxy to a Ext.data.Model using the following syntax:

```
proxy: {  
    type: 'localStorage',  
    id: 'someID'  
}
```

The following example illustrates defining a model for contacts that references a localstorage area named *localContacts*:

```
var Contact = Ext.regModel ('Contact',
{
    fields: [
        {name: 'lname', type: 'string', allowBlank: false },
        {name: 'fname', type: 'string', allowBlank: false },
        {name: 'email', type: 'string', allowBlank: false}
    ],
    validations: [
        {type: 'presence', field: 'fname'},
        {type: 'presence', field: 'lname'},
        {type: 'format', field: 'email', matcher: /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/}
    ],
    proxy: {
        type: 'localstorage',
        id: 'localContacts'
    }
})
```

## Putting Data into LocalStorage

Before you can put data into local storage you must link the model containing your localstorage proxy reference to an Ext.data.Store as indicated below:

```
var ContactStore = new Ext.data.Store({
    model: "Contact"
});
```

Once you have instantiated the Store, you can invoke the following methods to add, delete, and update data in localstorage:

Method	Description
add(Object data)	Adds Model instances to the Store by instantiating them based on a JavaScript object. The instances are added at the end of the collection. Accepts multiple data instances as arguments.
removeAt (number)	Removes the model instance at the given index
remove (Model / Array records)	Removes the given record from the store
sync()	Synchronizes the Store with its Proxy.

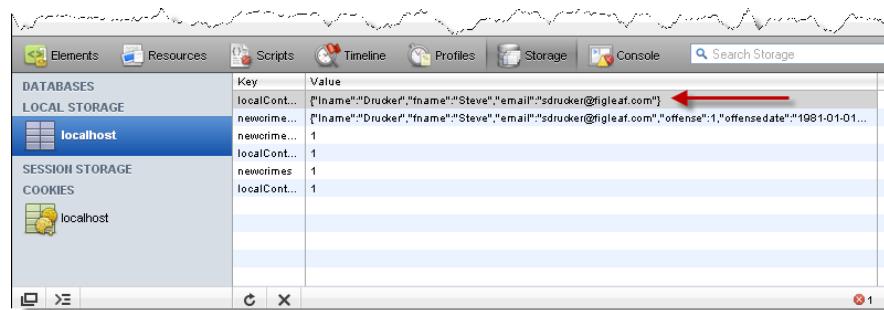
The following example illustrates placing a data record into the ContactStore and then writing the data to localstorage:

```
var contactData = Ext.ModelMgr.create(
{
    lname: 'Drucker',
    fname: 'Steve',
    email: 'sdrucker@figleaf.com'
}, Contact);

ContactStore.add(contactData);
ContactStore.sync();
```

## Verifying Content in LocalStorage

You can inspect the contents of your Local Storage repository by accessing the Storage tab of the debugging tools in Safari or Google Chrome. Note that abstract data types will be automatically serialized into JSON.



**Illustration 3: You can view the contents of the Local Storage repository**

## Reading Data from Local Storage

Use the following methods of Ext.data.Store to read from local storage:

Method	Description
load()	Loads data into the Store via the configured proxy.
each(function, scope)	Calls the specified function for each of the Records in the cache
first()	Returns the first data instance

Method	Description
getCount()	Gets the number of cached records. If using paging, this may not be the total size of the dataset.
getAt(Number index)	Get the Record at the specified index. This number is zero-based.

The following example illustrates loading data from local storage into a store and displays the contents of the fname field from the first data item:

```
var Contact= Ext.regModel ('Contact',
{
    fields: [
        {name: 'lname', type: 'string', allowBlank: false },
        {name: 'fname', type: 'string', allowBlank: false },
        {name: 'email', type: 'string', allowBlank: false}
    ],
    validations: [
        {type: 'presence', field: 'fname'},
        {type: 'presence',field: 'lname'},
        {type: 'format', field: 'email', matcher: /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/}
    ],
    proxy: {
        type: 'localstorage',
        id: 'localContacts'
    }
});

var ContactStore = new Ext.data.Store({
    model: "Contact"
});

ContactStore.load();

if (ContactStore.getCount() >= 1) {
    var firstDude = ContactStore.getAt(0);
    Ext.Msg.alert("Welcome",
                  firstDude.get('fname') + " was here");
}
```

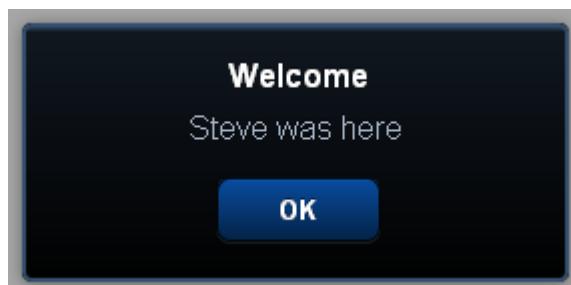


Illustration 4: Output of data retrieved from local storage

## Walkthrough 8-2: Caching Data on the Client



In this walkthrough, you will perform the following tasks:

- Cache web service data (Missing Children data) into localstorage
- Verify that data has been saved into localstorage
- Pull data from local storage if the application loses network connectivity

### Steps

#### Transfer Data to Local Storage

1. Open `/walk/walk8-2/index.html` in your editor
2. Review the `missingKidsStore` definition on lines 339-350. Note that the `autoLoad` property has been deleted.
3. Where indicated by the comment, insert an **IF** block that checks for network connectivity and whether the variable `bUseLocalCache` is false as indicated below:

```
if (navigator.onLine && !bUseLocalCache) {  
}  
else {  
}  
}
```

4. Inside the if block, invoke the `load()` method of the `missingKidsStore` and designate a callback function as indicated below:

```
missingKidsStore.load({  
    scope: this,  
    callback: function(records, operation) {  
  
    }  
})
```

5. Inside the callback function, invoke `missingKidsStore.setProxy()` to reset the proxy to localstorage as indicated below:

```
missingKidsStore.setProxy({  
    type: 'localStorage',  
    id: 'localMissingKids'  
})
```

6. Immediately following the code that you inserted in the prior step, invoke the `sync()` method to transfer all of the data in the store to local storage.

```
missingKidsStore.sync()
```

- 
7. Inside the else { } block, reset the proxy of missingKidsStore to localstorage as indicated by the following example:

```
missingKidsStore.setProxy({  
    type: 'localStorage',  
    id: 'localMissingKids'  
});
```

8. Underneath the code that you inserted in the prior step, invoke **missingKidsStore.load()** to pull data from localstorage.
9. Save the file and browse
10. Open your debugger and click on the **Storage** view
11. Click on **Local Storage > localhost**. You should see your dataset of missing children.
12. Return to **index.htm** and set the **bUseLocalCache** variable on line 359 to **true**.
13. Save the file and browse. Your data is now loading from local storage instead of from the remote web service.

– End of Walkthrough --

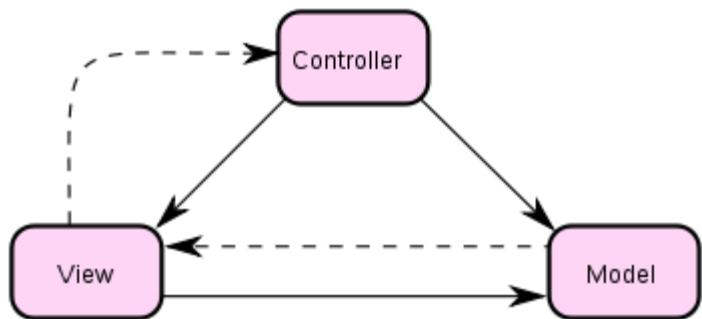
## Making your Code Base Manageable with SMVC



As you have already discovered, making code changes to your lab and walkthrough exercises becomes more challenging as the size of your application grows. Placing all of your JavaScript in a single file and not using object-oriented paradigms may work well for small, one-off applications, but over the long-term you'll want to use a framework to help maximize code re-use and to more quickly adjust to changing application specifications.

In order to facilitate code management, readability, and reuse, Sencha suggests that you implement a model-view-controller design pattern for your larger applications.

- The **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). In event-driven systems, the model notifies observers (usually views) when the information changes so that they can react. In Sencha Touch you define a model via the `Ext.regModel` constructor.
- The **view** renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. A viewport typically has a one to one correspondence with a display surface and knows how to render to it. In Sencha Touch, elements that inherit from the `Ext.Container` class are considered to be views.
- The **controller** receives input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input.<sup>1</sup> You can define a controller using the `Ext.Controller` class in Sencha Touch.



**Illustration 5:** The relationship between the model, view, and controller. Solid lines indicate direct connections.

*Note: There are many ways to implement an MVC architecture in Sencha Touch.. This unit introduces the concepts that underlie Sencha MVC (SMVC).*

<sup>1</sup> <http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller>

## Organizing your Files

In Sencha MVC, Models, Views, and Controllers are stored in separate files and located in a common directory structure. Using this structure helps facilitate coordination between multiple developers and share code between projects. While you may initially find this structure to be overly complex, note that you will use the Sencha Command command-line tool to automatically generate the skeletal structure of your new applications, their models, and their controllers.

### Defining your Application Root

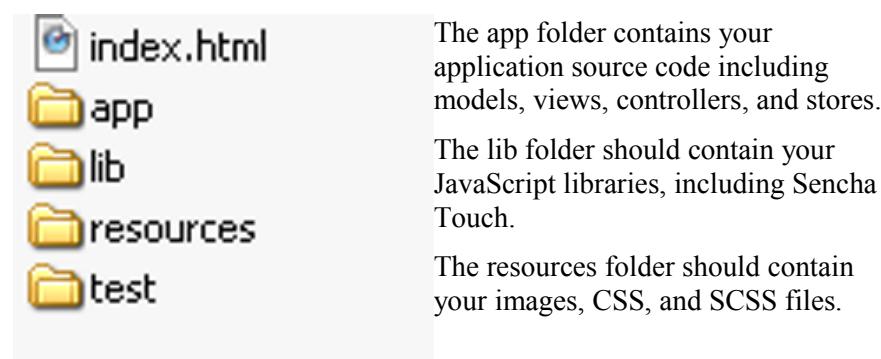


Illustration 6: SMVC Project Root

### Organizing your Application Source Code

Your `app` folder should contain directories that separate your models, views, controllers, and stores.

Your Sencha Touch application definition, initialized by the `Ext.regApplication()` method, should be placed in `app.js`.  
`routes.js` is where you should specify all of your application routes.

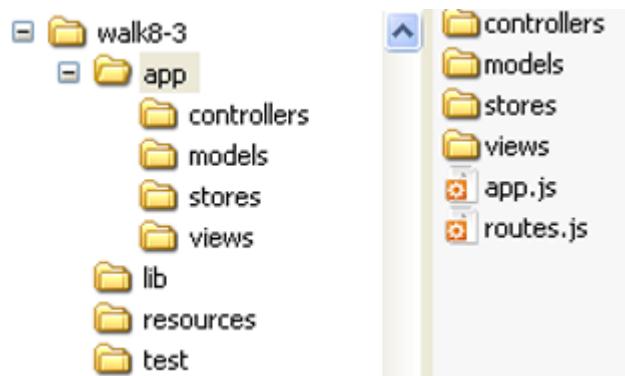
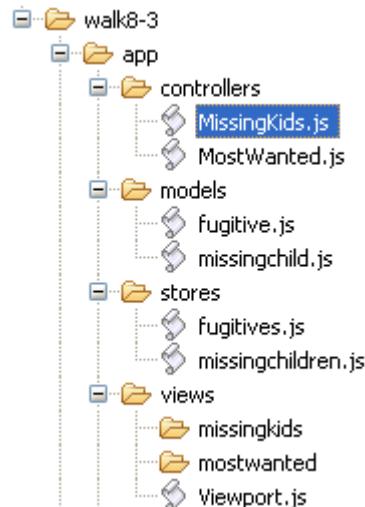


Illustration 7: SMVC app directory

## Understanding File Naming Conventions

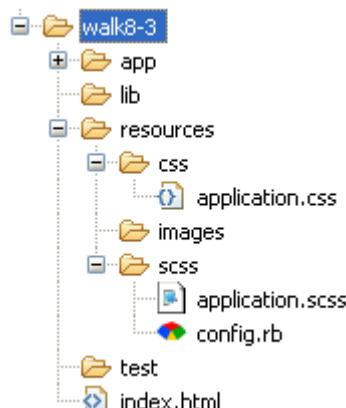
You can infer from the exploded file view, depicted at right, the following naming conventions:

- Models should be named for a singular object
- Stores should be named as the plural of the model.
- Views that are invoked by a controller should be placed in a subfolder that has the same name as their controller.



**Illustration 8: Organizing your MVC scripts**

## Organizing Your Resources

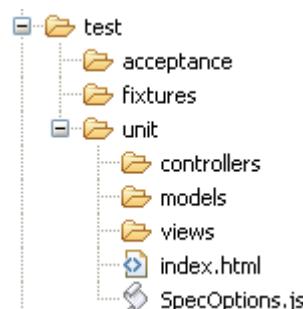


Your resources directory should resemble illustration 10, with your application.scss file generating the application.css file. Application image assets should be placed in the /images/ subfolder.

**Illustration 9: A typical SMVC resources files structure**

## Structuring your Unit Tests

The test folder should contain all of your unit test files for your models, views, and controllers.



**Illustration 10: A typical test folder**

## Using Sencha Command

Sencha Command is a command-line utility that automatically generates the SMVC file structure. Sencha Command is actually written in JavaScript, so you can open it up in an editor and review its functionality. You can run Sencha command from /senchatouch/jsbuilder/sencha.bat (Windows) or /senchatouch/jsbuilder/sencha.sh (Mac).

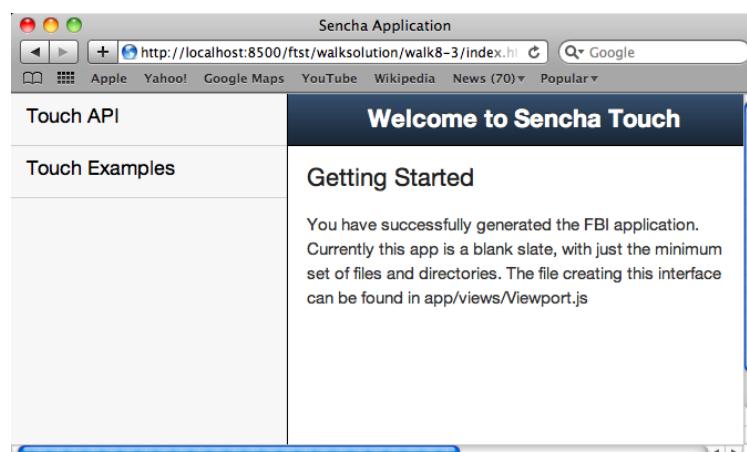
The command-line syntax for generating an application is the following:

```
sencha generate app [app name] [directory to store app]
```

```
*****  
Generating the fbi application  
*****  
  
Creating directories...  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/app  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/app/models  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/app/controllers  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/app/views  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/lib  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/public  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/public/resources/images  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/public/resources/css  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/acceptance  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/fixtures  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/unit  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/unit/models  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/unit/controllers  
Creating dir: C:\apache\htdocs\ftst\walk\walk8-3/test/unit/views  
Copying files...  
Copying index.html  
Copying app/routes.js  
Copying public/resources/css/application.css
```

**Illustration 11: Generating an application using Sencha Command**

Sencha Command generates a sample SMVC application, depicted below. Note that depending on the version of Sencha Touch that you are running, you may need to alter the script paths defined within the index.html file in order for the application to run.



**Illustration 12: The SMVC sample application**

The generated index.html file is listed below. Note the following:

- In order to use SMVC you must load the /pkgs/platform/mvc.js script. This is a component of the Sencha core.
- SMVC can also be applied to developing EXT-JS applications.
- Your model, view, and controllers must be instantiated at specific locations within the HTML file.

```
<!doctype html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Sencha Application</title>

    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0;
                    maximum-scale=1.0; user-scalable=0;" />

    <meta name="apple-mobile-web-app-capable" content="yes" />
    <link rel="apple-touch-icon" href="apple-touch-icon.png" />

    <link rel="stylesheet"
        href="/senchatouch/resources/css/sencha-touch.css"
        type="text/css">

    <link rel="stylesheet"
        href="public/resources/css/application.css" type="text/css">
</head>
<body>
    <script type="text/javascript"
        src="senchatouch/sencha-touch-debug.js"></script>
    <script type="text/javascript"
        src="/senchatouch/pkgs/platform/mvc.js"></script>

    <div id="sencha-app">
        <script type="text/javascript" src="app/routes.js"></script>
        <script type="text/javascript" src="app/app.js"></script>

        <!-- Place your view files here -->
        <div id="sencha-views">
            <script type="text/javascript" src="app/views/Viewport.js"></script>
        </div>

        <!-- Place your model files here -->
        <div id="sencha-models">

        </div>

        <!-- Place your controller files here -->
        <div id="sencha-controllers">

        </div>
    </div>
</body>
</html>
```

## Coding in SMVC

Sencha Touch ships with the following Manager classes:

- Ext.regApplication() creates a new Application class from the specified configuration object.
- Ext.regModel, as you learned about earlier in this class, registers a new data model and optionally binds it to a proxy.
- Ext.regStore(), covered in unit 5, defines a data store that exposes your models to be globally accessible from within your application's controllers.
- Ext.regController() creates a controller class from the specified configuration object.
- Ext.reg() is used to register new xtype's.
- Ext.dispatch() is used to execute different controller actions

### Defining Applications with Ext.regApplication()

Ext.regApplication() is a pointer to the Ext.Application class in Sencha Touch, representing a Sencha Application. Most applications consist of an application name and launch function as illustrated below. Applications are defined within the app/app.js file.

```
Ext.regApplication({

    defaultTarget: 'viewport',
    name: 'FBI'
    useHistory:true,
    defaultUrl: 'MissingKid/index',

    tabletStartupScreen: '../../../../../images/cf_tablet_startup.png',
    phoneStartupScreen: '../../../../../images/cf_phone_startup.png',
    icon: '../../../../../ftst/images/cf_phone_icon.png',
    glossOnIcon: false,

    launch: function() {
        this.viewport = new FBI.Viewport({application: this})
    }
})
```

Note the following;

- SMVC has history support and deep linking via the `useHistory` and `defaultUrl` configuration attributes.
- The `defaultUrl` routes to the `index` method of the `MissingKid` controller
- The `launch()` function will only execute once.

## Defining Models

You can use Sencha Command to generate your data Models by using the following syntax from the root of your project directory:

```
sencha generate model [model name] [fieldN:typeN]
```

For example, to define a Model named missingkid containing four string fields – TITLE, FIRSTNAME, LASTNAME, and PHOTO you would use the following syntax:

```
sencha generate model missingkid TITLE:string
FIRSTNAME:string LASTNAME:string PHOTO:string
```

Generating a model using this method results in the addition of three new files to your project:

File	Description
app/models/missingkid.js	The model, defined using the Ext.regModel() method.
test/unit/models/missingkid.js	Invokes the model and defines an empty instance.
test/fixtures/missingkid.js	Defines an empty fixture for the model

*Note: Sencha command also adds a <script> reference to the app/models/missingkid.js to your index.html file.*

The resulting /app/models/person.js file resembles the following:

```
Ext.regModel("missingkid", {
    fields: [
        {name: "TITLE", type: "string"},
        {name: "FIRSTNAME", type: "string"},
        {name: "LASTNAME", type: "string"},
        {name: "PHOTO", type: "string"}
    ]
});
```

Once Sencha Command has defined your model you will likely need to add the following items to it manually:

- A proxy definition
- Validations
- Associations

The following examples illustrates the completed model definition that is used to retrieve structured data from a remote server:

```
Ext.regModel("missingkid", {
    fields: [
        {name: "TITLE", type: "string"}, 
        {name: "FIRSTNAME", type: "string"}, 
        {name: "LASTNAME", type: "string"}, 
        {name: "DESCRIPTION", type: "string"}, 
        {name: "PHOTO", type: "string"} 
    ],
    proxy: {
        type: 'scripttag',
        url: 
            'http://www.senchatouchtraining.com/ftst/components/crimeservice.cfc?method=getmissingchildrenjsonp',
        reader: {
            type: 'json'
        }
    }
});
```

## Defining Stores

Stores should be placed in the app/stores folder. Their file names should be the plural of their bound model names. You will need to build these files manually.

A typical store might resemble the following:

```
Ext.regStore('MissingKidsStore', {
    model: 'missingkid',
    autoload: true,
    sorters: ['LASTNAME'],
    getGroupString: function(record) {
        return record.get('LASTNAME')[0];
    }
})
```

Currently there is no Sencha Touch method to explicitly register a Tree Store. Tree stores are registered implicitly using the `storeId` configuration attribute of the `Ext.data.TreeStore` constructor as illustrated below:

```
MostWantedStore = new Ext.data.TreeStore({
    model: 'mostwanted',
    storeId: 'MostWantedStore',
    autoLoad: true
});
```

*Note: You will need to manually add your Store definitions to your index.html file.*

## Defining Views

You will create views within your project inside the app/views folder. Each unique view should be placed within its own subfolder. Typically, Views have the following characteristics:

- They extend a Sencha Touch component
- They optionally reference a data store
- They register themselves as a custom xtype

The following example illustrates defining the FBI Missing Kids panel using SMVC:

```
FBI.views.MKDetailPanel = Ext.extend(Ext.Panel, {
    layout: 'fit',
    initComponent: function() {
        this.store = Ext.getStore('MissingKidsStore');
        this.list= new Ext.List({
            itemTpl: '<div class="contact">{TITLE}</div>',
            store : this.store,
            grouped : true,
            indexBar : true,
            pinHeaders : true
        });
        this.items = [this.list];
    }
});
Ext.reg('fbi-missingkidpanel',FBI.views.MKDetailPanel);
```

The following example illustrates a second view which is invoked when a user taps on an item in the MissingKidPanel list. Note that the logic for jumping between views is handled by the controller.

```
FBI.views.MKDetailPanel = Ext.extend(Ext.Panel, {
    tpl: '<div class="contact">{DESCRIPTION}</div>',
    initComponent: function() {
        this.dockedItems = {
            dock: 'top', xtype: 'toolbar',
            id: 'missingKidDetailToolbar', title: 'Placeholder',
            items: [
                {text: 'Back', ui: 'back', itemId: 'missingKidDetailBackButton'}
            ]
        };
    }
});
Ext.reg('fbi-missingkid-detail',FBI.views.MKDetailPanel);
```

## Defining Controllers

You can use Sencha Command to generate your data Controllers by using the following syntax from the root of your project directory:

```
sencha generate controller [Controller Name] [actions]
```

In order to generate a controller for the MissingKids subsystem that supported two actions – index and show, you would use the following command-line syntax from the root of your project:

```
sencha generate controller MissingKid index show
```

Generating a controller using this method results in the addition of two new files to your project:

File	Description
app/controllers/missingkid.js	An empty controller, instantiated using the Ext.regController() method
test/unit/controllers/missingkid.js	Invokes the model and defines an empty instance.

*Note: Sencha command also adds a <script> reference to the app/controllers/missingkid.js to your index.html file*

The resulting app/controllers/missingkid.js file contains the following code:

```
Ext.regController("MissingKid", {  
    index: function() { },  
  
    show: function() { }  
});
```

Typically, your Controller will perform the following functions:

- Instantiate views
- Define event listeners for user actions
- Destroy views that are not currently active, thereby reducing DOM overhead
- Apply transitions between views

The following code example illustrates a typical controller:

```
Ext.regController("MissingKid", {

    index: function() {
        if (!this.listPanel) {

            this.listPanel = this.render({
                xtype: 'fbi-missingkidpanel',
                listeners: {
                    list: {
                        select: this.show, // user selected item in list
                        scope: this // so jump to show() action
                    },
                    activate: function(listPanel) {
                        listPanel.list.getSelectionModel().deselectAll();
                    }
                }
            });

            this.application.viewport.setActiveItem(this.listPanel);
        } else {

            this.application.viewport.setActiveItem(this.listPanel, {
                type: 'slide',
                direction: 'right'
            });
        }
    }

} // end of index action,

show: function(list, record) {
    var detailPanel = this.render({
        xtype: 'fbi-missingkid-detail',
        data: record.data,
        listeners: {
            deactivate: function (detailPanel) {
                detailPanel.destroy();
            }
        }
    });

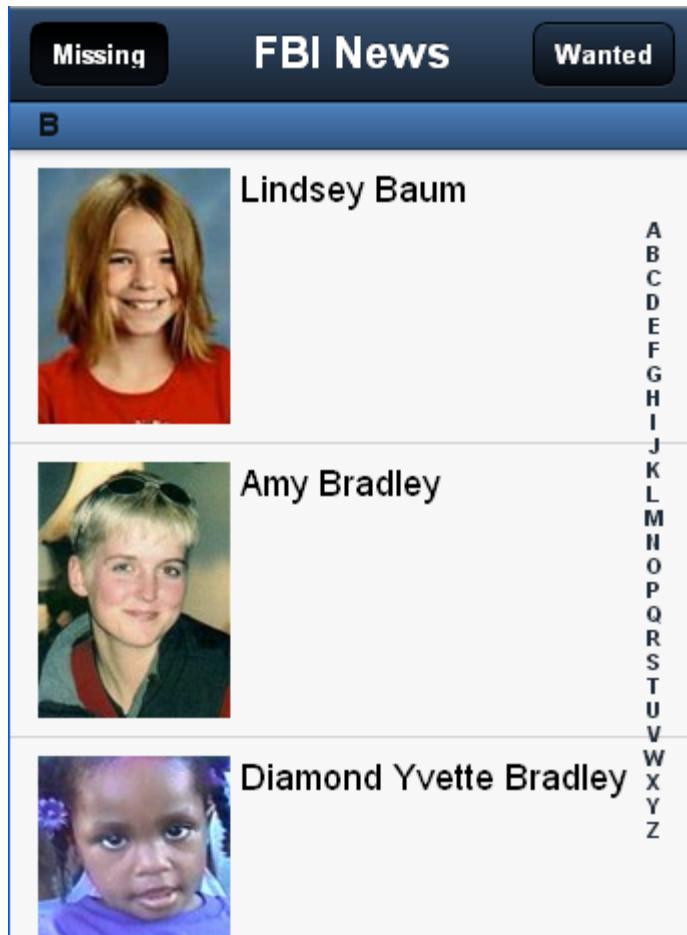
    // add event listener to back button, located on view
    detailPanel.query("#missingKidDetailBackButton")[0].on({
        tap: this.index, // jump back to index() action
        scope: this
    })

    // Change title on view
    Ext.getCmp('missingKidDetailToolbar').setTitle(record.get("FIRSTNAME") + " " + record.get("LASTNAME"));

    // Display view panel
    this.application.viewport.setActiveItem(detailPanel, {
        type: 'slide',
        direction: 'left'
    })
} // end of show()
});
```

## Routing Between Controllers in the Viewport with Ext.Dispatch()

Most of your applications will need to route between multiple controllers. For example, the FBI News application, depicted below, has buttons that allow the user to navigate between a system that displays a list of missing children view and a system that displays the ten-most wanted fugitives.



**Illustration 13: The FBI News SMVC Application**

Use `Ext.Dispatch()` in your button handlers to change the contents of the Viewport as illustrated below:

```
Ext.dispatch({
    controller: 'TenMostWanted',
    action     : 'index',
    historyUrl: 'TenMostWanted/index'
});
```

The following code listing of `Viewport.js` fully illustrates how transferring execution between controllers can be achieved.

```
FBI.Viewport = Ext.extend(Ext.Panel, {
    id         : 'viewport',
    layout     : 'card',
    fullscreen : true,
    initComponent: function() {
        Ext.apply(this, {
            dockedItems: [
                {
                    dock : 'top',
                    xtype: 'toolbar',
                    title: 'FBI News',
                    layout: {pack: 'center'},
                    items: [
                        {
                            xtype: 'button',
                            id: 'btnMissing',
                            text: 'Missing',
                            componentCls: 'x-button-pressed',
                            handler: this.onKidsTap,
                            scope: this
                        },
                        {xtype: 'spacer'},
                        {
                            xtype: 'button',
                            id: 'btnWanted',
                            text: 'Wanted',
                            handler: this.onWantedTap,
                            scope: this
                        }
                    ]
                }],
            });
        }); // Ext.apply

        FBI.Viewport.superclass.initComponent.apply(this,
  arguments);
    }, // initComponent

    onKidsTap: function(btn) {
        Ext.getCmp('btnWanted').removeCls("x-button-pressed");
        btn.addCls("x-button-pressed");
        Ext.dispatch({
            controller: 'MissingKid',
            action     : 'index',
            historyUrl: 'MissingKid/index'
        });
    },

    onWantedTap: function(btn) {
        Ext.getCmp('btnMissing').removeCls("x-button-pressed");
        btn.addCls("x-button-pressed");
        Ext.dispatch({
            controller: 'TenMostWanted',
            action     : 'index',
            historyUrl: 'TenMostWanted/index'
        });
    }
});
```

## Walkthrough 8-3: Working with Sencha MVC



In this walkthrough, you will perform the following tasks:

- Define a Model, View, and Controller in Sencha MVC
- Link the index.html file to the MVC
- Modify your viewport to enable toggling between views

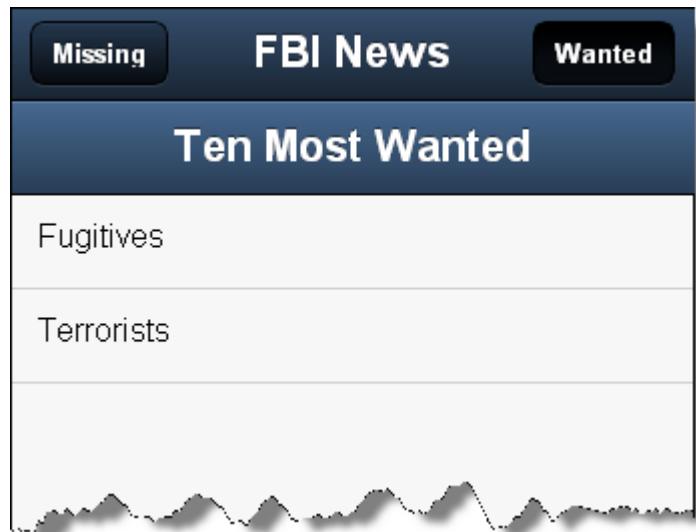


Illustration 14: You will refactor the Ten Most Wanted Tree List into SMVC

## Steps

### Review the Codebase

1. Open [/walk8-3/](#) and review the codebase with your instructor.
2. Browse [/walk8-3/app/index.html](#) to view the existing application. You will add the Ten Most Wanted MVC components.

### Define the Model

3. In `/walk8-3/app/models/`, create a new, empty file named `mostwanted.js`

4. Register a new model with the following characteristics:
  - Model Name: mostwanted
  - Fields: name: text, type: string
  - Proxy: Reads tree-structured data from  
<http://www.senchatouchtraining.com/ftst/components/crimeservice.cfc?method=getoptenjsonp>

5. Verify that your code appears similar to the following:

```
Ext.regModel("mostwanted", {
    fields: [
        {name: 'text', type: 'string'}
    ],
    proxy: {
        type: 'scripttag',
        url :
        'http://www.senchatouchtraining.com/ftst/components/crimeservice.cfc?method=getoptenjsonp',
        reader: {
            type: 'tree',
            root: 'items'
        }
    }
});
```

6. Save the file
7. Open /walk8-3/index.html and add a <script> tag at the appropriate location that links to your model.
8. Save the file

### Define the Store

9. In /walk8-3/app/stores/, create a new, empty file named **mostwantedlists.js**
10. Inside the file, define a new Tree store named **MostWantedStore** that references the model you created in step 4. Your code should appear as follows:

```
MostWantedStore = new Ext.data.TreeStore({
    model: 'mostwanted',
    storeId: 'MostWantedStore',
    autoLoad: true
});
```

11. Save the file
12. Open /walk8-3/index.html and add a <script> tag at the appropriate location that links to your store.
13. Save the file

### Define the View

14. Create a new subfolder of /walk8-3/app/views named **mostwanted**
15. In /walk8-3/app/views/mostwanted/, create a new empty file named **MostWantedTreeListPanel.js**

16. Inside the file, extend **Ext.Panel** with an **Ext.NestedList** that outputs the contents of **MostWantedStore**. Your code should appear as follows:

```
FBI.views.TenMostWantedPanel = Ext.extend(Ext.Panel, {
    layout: 'fit',
    initComponent: function() {
        this.store = Ext.getStore('MostWantedStore');

        this.treeList= new Ext.NestedList({
            store: this.store,
            title: 'Ten Most Wanted',
            displayField: 'text',
        })
        this.items = [this.treeList];
    }
})
FBI.views.TenMostWantedPanel.superclass.initComponent.apply(this,arguments);
})
```

17. At the bottom of the file, register your view as an xtype named **fbi-tenmostwantedpanel** as illustrated below:

```
Ext.reg('fbi-tenmostwantedpanel',
FBI.views.TenMostWantedPanel);
```

18. Save the file

19. Return to /walk8-3/index.html and add a <script> tag at the appropriate location that links to your view.

### Define the Controller

20. In /walk8-3/app/controllers/, create a new empty file named **MostWanted.js**

21. Inside the file, register a controller named **TenMostWanted**

22. Inside the controller, define a single action named **index**

23. Inside the action, insert code to render the xtype **fbi-tenmostwantedpanel**

24. Verify that your controller appears similar to the following:

```
Ext.regController("TenMostWanted", {
    index: function() {
        if (!this.treeViewPanel) {
            this.treeViewPanel = this.render({ xtype: 'fbi-tenmostwantedpanel' })
            this.application.viewport.setActiveItem(this.treeViewPanel);
        } else {
            this.application.viewport.setActiveItem(
                this.treeViewPanel, { type: 'slide', direction: 'left' })
        }
    }
});
```

25. Save the file

26. Return to /walk8-3/index.html and add a <script> tag at the appropriate location that links to your controller.
27. Save the file

#### **Link the Most Wanted subsystem through the Viewport**

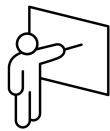
28. Open /walk8-3/views/Viewport.js and review the code with your instructor.
29. Inside the **onWantedTap()** method, invoke a method to remove the **x-button-pressed** class from the **btnMissing** button.
30. Underneath the code that you inserted in the prior step, insert a method call to add the **x-button-pressed** CSS class to the currently pressed button.
31. Insert a command to invoke the **TenMostWanted index** action.
32. Save the file

#### **Test your work**

33. Return to /walk8-3/index.html
34. Browse the file.
35. Click on the Wanted button and verify that the Ten Most Wanted lists appear.

– End of Walkthrough –

# Optimizing your Distribution with JSBuilder



JSBuilder is a multi-OS compatible, command-line JavaScript and CSS project build tool that combines and minifies your source files resulting in a distribution that downloads in less time to your device and compiles faster in its browser. JSBuilder ships with Sencha Touch and is located within its /jsbuilder subdirectory.

## Creating a Project File

JSBuilder project files (.jsb3) contain JSON encoded directives for managing your builds. For example, the .jsb3 file used to generate sencha-touch.js is located at /senchatouch/sencha-touch.jsb3. You can modify this jsb3 file in order to generate a smaller sencha-touch.js file that excludes any components that you are not using in your application. Currently there is no GUI for creating .jsb3 files.

Invoke JSBuilder using the following syntax:

```
jsbuilder --projectfile.jsb3
```

The top-level of the JSB3 file should contain the following attributes:

Attribute	Description
projectName	String describing the project
licenseText	String specifying the header of all .js and .css. Use \n for newlines
pkgs	An array of package descriptors
builds	An array of build descriptors
resources	An array or resource descriptors.

Therefore, a typical .jsb3 file structure would resemble the following:

```
{
  "projectName": "CrimeFinder",
  "licenseText": "Copyright(c) 2011 Sencha",
  "packages": [ ],
  "builds" : [ ],
  "resources": [ ]
}
```

```

"projectName": "Sencha Touch",
"licenseText": "Copyright(c) 2010 Sencha Inc.\nlicensing@sencha.com\nhttp://www.sencha.com/touchlicense",
"packages": [
    {
        "name" : "Platform Util",
        "target": "pkgs/platform/util.js",
        "id"   : "platform-util",
        "files": [
            {"path": "src/platform/src/", "name": "Ext.js"},  

            {"path": "src/platform/src/util/", "name": "Observable.js"},  

            {"path": "src/platform/src/util/", "name": "Stateful.js"},  

            {"path": "src/platform/src/util/", "name": "HashMap.js"},  

            {"path": "src/platform/src/util/", "name": "MixedCollection.js"},  

            {"path": "src/platform/src/util/", "name": "AbstractManager.js"},  

            {"path": "src/platform/src/util/", "name": "DelayedTask.js"},  

            {"path": "src/platform/src/util/", "name": "GeoLocation.js"},  

            {"path": "src/platform/src/util/", "name": "Point.js"},  

            {"path": "src/platform/src/util/", "name": "Offset.js"},  

            {"path": "src/platform/src/util/", "name": "Region.js"},  

            {"path": "src/platform/src/util/", "name": "Template.js"},  

            {"path": "src/platform/src/util/", "name": "XTemplate.js"},  

            {"path": "src/platform/src/util/", "name": "Sorter.js"},  

            {"path": "src/platform/src/util/", "name": "Filter.js"},  

            {"path": "src/platform/src/util/", "name": "Function.js"},  

            {"path": "src/platform/src/util/", "name": "Date.js"},  

            {"path": "src/platform/src/util/", "name": "Number.js"},  

            {"path": "src/platform/src/util/", "name": "Format.js"},  

            {"path": "src/platform/src/util/", "name": "LoadMask.js"}
        ]
    },
    {
        "name" : "Platform Native",
        "target": "pkgs/platform/native.js",
        "id"   : "platform-native",
        "files": [
            {"path": "src/platform/src/native/", "name": "Array.js"}
        ]
    },
    {
        "name" : "Platform Core",
        "target": "pkgs/platform/core.js",
        "id"   : "platform-core",
        "files": [
            {"path": "src/platform/src/", "name": "ComponentMgr.js"},  

            {"path": "src/platform/src/", "name": "ComponentQuery.js"},  

            {"path": "src/platform/src/", "name": "PluginMgr.js"},  

            {"path": "src/platform/src/", "name": "EventManager.js"},  

            {"path": "src/platform/src/", "name": "Support.js"}
        ]
    }
]

```

**Illustration 15: The Sencha Touch JSB3 file**

## Describing a Package

Package descriptors use the following attributes:

Attribute	Description
name	String describing the package
target	String specifying the file to create
id	String that acts as a unique identifier. Specified within the “builds” section and used to combine packages into a single file.
files	An array of files to be minified and combined. Each entry in the array has a path attribute and a name attribute.

The following example illustrates a valid package definition:

```
{
  "name" : "Platform Core",
  "target": "pkgs/platform/core.js",
  "id"   : "platform-core",
  "files" : [
    {"path": "src/platform/src/", "name": "ComponentMgr.js"},
    {"path": "src/platform/src/", "name": "Support.js"}
  ]
}
```

## Describing Builds

Builds are used to merge packages into a single .js file. Create your build definitions using the following structure:

Attribute	Description
name	A string representing the build name
target	A string representing the file to be output
debug	A boolean representing whether to create a debug build or an uncompressed file.
packages	Array of package identifiers.

The following example illustrates a valid build definition:

```
"builds": [
  {
    "name": "Sencha Touch",
    "target": "sencha-touch.js",
    "debug": true,
    "options": {
      "minVersion": 1.1
    },
    "packages": [
      "platform-util",
      "platform-native",
      "platform-core",
      "layouts"
    ]
  }
]
```

## Describing Resources

The resources section contains entries that result in files being moved into your project. Resource descriptors use the following attributes:

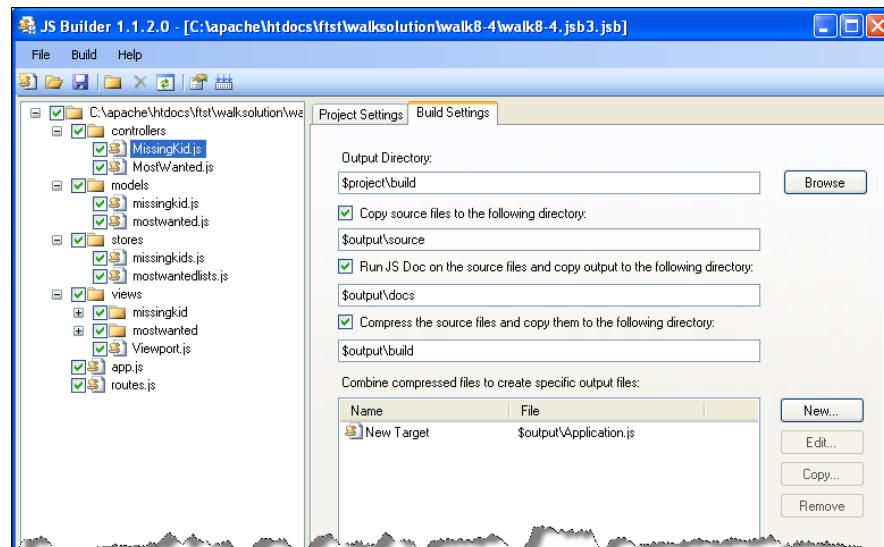
Attribute	Description
src	String describing the folder to move resources from
dest	String describing the folder to move resources to

The following example illustrates a resource definition:

```
"resources": [
    {"src": "resources/"},
    {"src": "sencha-touch.jsb3"},
    {"src": "getting-started.html"},
    {"src": "index.html"}
]
```

## Using the JSBuilder 1.x GUI

Manually editing a JSON file can be a daunting task. While there currently is no GUI available for JSB3 files, there does exist a Windows-compatible GUI for JSB 1 files, depicted below. You can download this application at <http://code.google.com/p/js-builder/>



**Illustration 16: The JSBuilder 1 GUI for Windows**

*Note: JSB1 files are not compatible with the version of JSBuilder that ships with Sencha Touch. However, you can still use it to build one-off distributions.*

## Ordering your Merge Files

Due to idiosyncrasies in how mobile browsers download, compile, and execute scripts you should merge all of your Javascripts into a single file in order to achieve optimal performance. When using JSBuilder to combine scripts developed with SMVC, list the files in the following order:

1. /senchatouch/sencha-touch.js
2. /senchatouch/pkgs/platform/mvc.js
3. app/routes.js
4. app/app.js
5. app/views/Viewport.js
6. app/views/\*.js
7. app/models/\*.js
8. app/stores/\*.js
9. app/controllers/\*.js

## Running JSBuilder

Use the command-line version of JSBuilder to script the deployment of your application. It accepts the following arguments:

Argument	Shortcut	Description
--projectFile	-p	Required. Location of a jsb2 project file
--homeDir	-d	Required. Home directory to build the project to
--verbose	-v	Optional. Output detailed information about what is being built
--debugSuffix	-s	Optional. Suffix to append to JS debug targets, defaults to 'debug'
--help	-h	Optional. Prints this help display.

The following examples illustrate sample usage:

```
JSBuilder -p myapp.jsb -d C:\Apps\www\deploy\
```

## Modifying your deployment index.html file

Once you have all of your scripts merged and minified into a single file, you will need to modify your index.html file. A typical SMVC index.html production file should resemble the following (assuming that you used Compass and SASS to generate a single optimized CSS file):

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Sencha Application</title>

    <meta name="viewport" content="width=device-width,
user-scalable=no, initial-scale=1.0; maximum-scale=1.0;
user-scalable=0;" />
    <meta name="apple-mobile-web-app-capable"
content="yes" />
    <link rel="apple-touch-icon" href="apple-touch-
icon.png" />

    <link rel="stylesheet"
        href="Application.css" type="text/css">

    <script type="text/javascript"
        src="Application.js"></script>
</head>
<body>

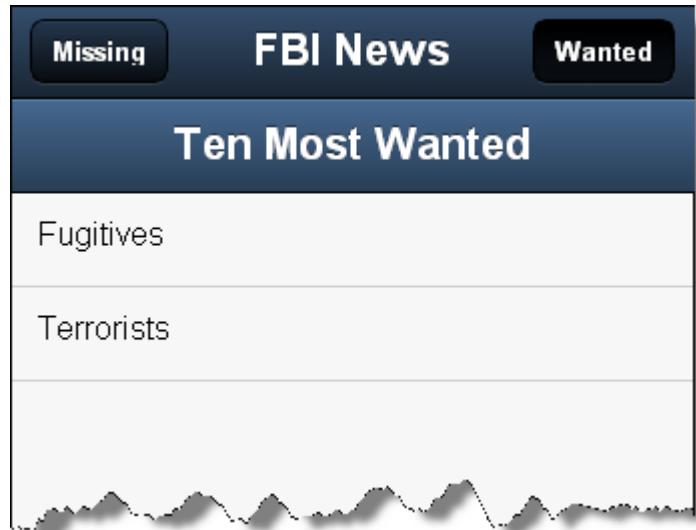
</body>
</html>
```

## Demonstration 8-4: Optimizing for Production



In this walkthrough, you will perform the following tasks:

- Use JSBuilder to define a .jsb file
- Merge your Javascripts into a single, minfied file
- Modify your index.html for production



**Illustration 17: You will minify the codebase of your SMVC application**

### Steps

#### Select your JavaScript Files

1. Download and install JSBuilder from <http://code.google.com/p/js-builder/>.
2. Run JSBuilder
3. On the JSBuilder menu bar, select **File > New Project**
4. Enter the following information:
  - Project Name: **walk8-4**
  - File Name: **c:\apache\htdocs\ftst\walk\walk8-4\walk8-4.jsb**
5. Click **OK**
6. Modify the entries in the **Project Settings** tab as you see fit
7. Click on the **Add Folder** button, located on the JS Builder button bar

8. Select the folder **/walk8-4/app**
9. Click the checkbox adjacent to the **/walk8-4/app** directory

#### **Merge your JavaScript Files**

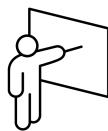
10. Click on the **Build Settings** tab
11. Set the name field to **Application.js**
12. Change the **Output File name** to **\$output\Application.js**
13. Click on the **New** button to Combine compressed files to create specific output files
14. Drag the **Project Files** into the **Included Files** box in the following order. Note that you will be able to reorder the files after they have been dragged.
  1. app/routes.js
  2. app/app.js
  3. app/views/Viewport.js
  4. app/views/\*.js
  5. app/models/\*.js
  6. app/stores/\*.js
  7. app/controllers/\*.js
15. Click **OK**
16. Select **File > Save Project**
17. Select **Build > Build Project**
18. Click **OK** to ignore the error message regarding JSDoc
19. Return to Aptana Studio and refresh your project
20. Open the **/walk8-4/build/Application.js** file and review its contents

#### **Adapt your index.html file**

21. In Aptana, open **/walk8-4/index.html**
22. Delete all of the code between the **<body>...</body>** tags except for the references to **sencha-touch.js** and **mvc.js**
23. Underneath the **<script>** tag that points to **mvc.js**, insert a **<script>** tag that points to **build/Application.js**
24. Save the file and test.

– End of Walkthrough –

# Compiling Native Apps with PhoneGap



PhoneGap is a third-party, open source development framework for building cross platform mobile apps. By combining it with Sencha Touch you can turn your iOS and Android web applications into native apps that can be distributed through the Android MarketPlace or iPhone App Store.

PhoneGap enables you to extend the capabilities of your application by exposing your device's local hardware resources via a JavaScript API. Supported resources include the following:

- Accelerometer
- Contacts
- Compass
- File System
- Camera
- Device Notifications
- Custom Events

The screenshot shows the official PhoneGap website. At the top, there is a navigation bar with links for Docs, Community, About, Tools, Blog, and Services. Below the navigation is a large "Get Started" button. To the right of the button are two orange buttons labeled "Download PhoneGap" and "Get Started". The main content area features a grid of mobile phone icons, each with a corresponding platform name. The platforms listed are:

Platform	Icon
iOS (Xcode)	
Windows Mobile	
Android (Eclipse)	
Symbian (Nokia WRT)	
Android (Terminal)	
Symbian (Sony Ericsson)	
Blackberry Widgets (5+)	
Symbian (Qt)	
Blackberry 4.x	
webOS	

**Illustration 18: PhoneGap enables you to create native applications for a wide variety of mobile devices**

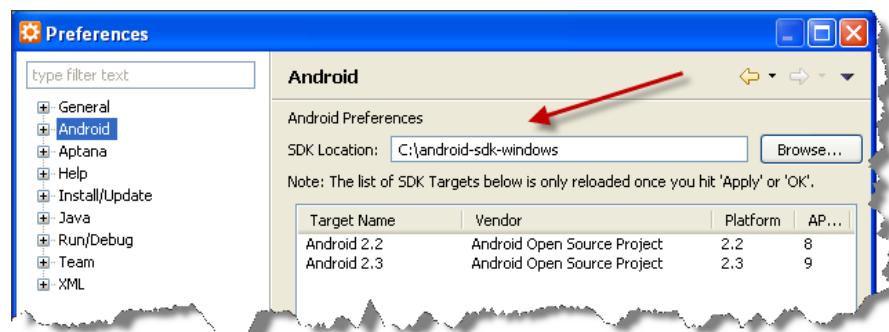
## Getting Started with PhoneGap for Android

Working with PhoneGap for Android requires that you have the following software installed on your development workstation:

- The Android SDK with at least one virtual device defined (described in unit 2)  
<http://developer.android.com/sdk>
- Android Developer Tools (ADT) for Eclipse  
<http://developer.android.com/sdk/eclipse-adt.html>

*Note: Android Developer Tools (ADT) for Eclipse is compatible with Aptana 2.*

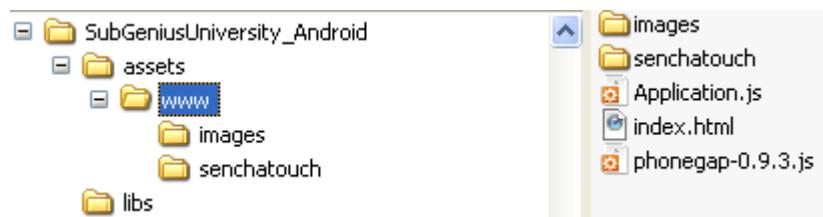
Once your ADT is installed you will need to point it to your Android SDK from the Aptana Preferences dialog box as depicted below:



**Illustration 19: Configuring the ADT for Eclipse**

## Organizing your Sencha Codebase

The first step in transforming your Sencha Touch application into a native app is to reorganize its directory structure. All of your application resources including your sencha-touch.js and sencha-touch.css files should be placed into an assets/www folder as indicated by the following screenshot. Don't forget to include any image, video, and audio assets that you would also like to host on the device. You will need to manually create the /assets/www and /libs folder.



**Illustration 20: Reorganizing your application file structure for PhoneGap**

## Adding PhoneGap Files to your Sencha Codebase

Once you have reorganized your project you will need to copy the following files from your PhoneGap SDK into the listed subdirectories of your project:

PhoneGap File	Project Directory
Android/phonegap.jar	/libs
Android/phonegap.js	/assets/www

Once the files are in place you will need to load the phonegap.js file into your index.html file if you plan to use any of the PhoneGap API methods, such as camera capture. You will also need to wrap your Ext.setup() method with with an event listener for PhoneGap's deviceready event as indicated by the code sample below:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>CrimeFinder</title>

<link rel="stylesheet"
      href="senchatouch/sencha-touch.css"
      type="text/css" />

<script type="text/javascript"
       src="senchatouch/sencha-touch.js"></script>

<script type="text/javascript"
       src="http://maps.google.com/maps/api/js?sensor=true">
</script>

</head>
<body>

<script type="text/javascript"
       charset="utf-8" src="phonegap-0.9.3.js"></script>

<script type="text/javascript">
document.addEventListener("deviceready", function() {
    Ext.setup({
        tabletStartupScreen: 'images/cf_tablet_startup.png',
        phoneStartupScreen: 'images/cf_phone_startup.png',
        icon: 'images/cf_phone_icon.png',
        glossOnIcon: false,
        onReady: function() {
            // your code here
        }
    })
</script>

</body>
</html>
```

## Defining a New Android Project

Once you have installed the ADT and reorganized your files you can define a new Android project as illustrated below:



**Illustration 21: Using the ADT Plugin for Eclipse**

The resulting dialog, displayed on the following page, prompts you for the following information:

Field	Description
<b>Project Name</b>	The Eclipse reference to the project
<b>Contents</b>	Location of your reorganized source files
<b>Build Target</b>	The version of Android that your application is targeting. You should always select the latest version. Phonegap will handle backwards compatibility
<b>Application name</b>	The name of your application
<b>Package name</b>	You can use virtually any package name, however, the best practice is to set this as com.phonegap. <i>appname</i> where <i>appname</i> is your application name.
<b>Create Activity</b>	This should be set to <i>App</i>
<b>Min SDK Version</b>	(Leave empty)

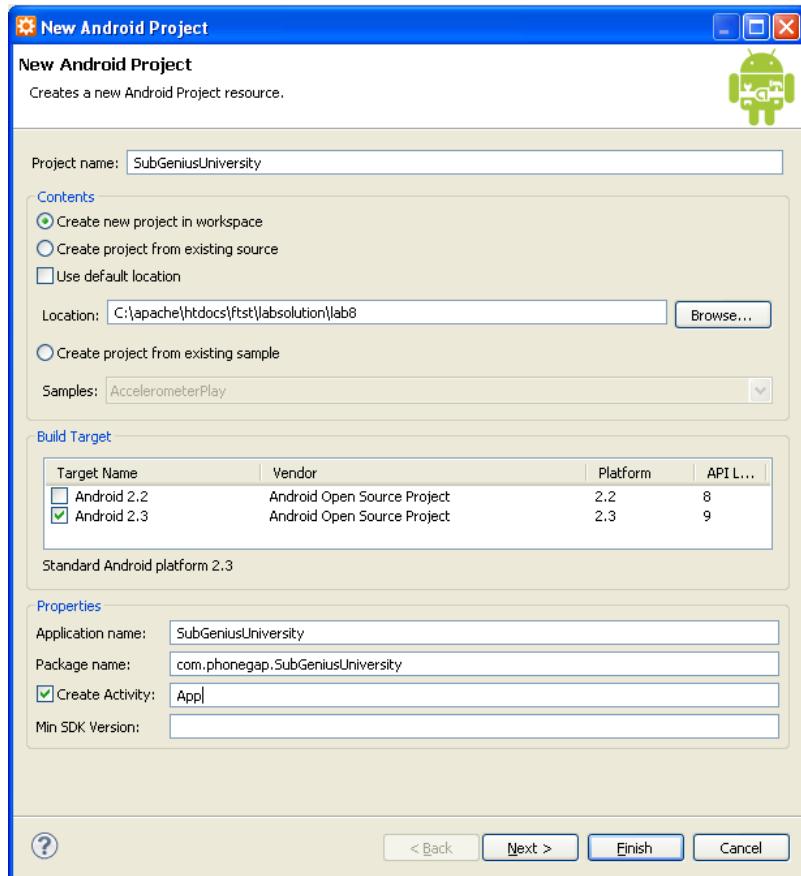
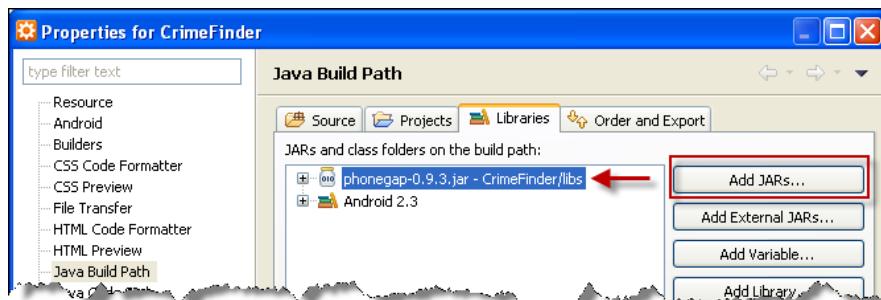


Illustration 22: Defining a new Android Project

## Adding PhoneGap.jar to your build path

Once your project files have been generated you must add the phonegap.jar file to your build path. You can access this feature by right-clicking on your libs folder in the project explorer and select Build Paths > Configure Build Paths. From the dialog depicted below you can add phonegap.jar to your project.



## Modifying the Generated Java Source

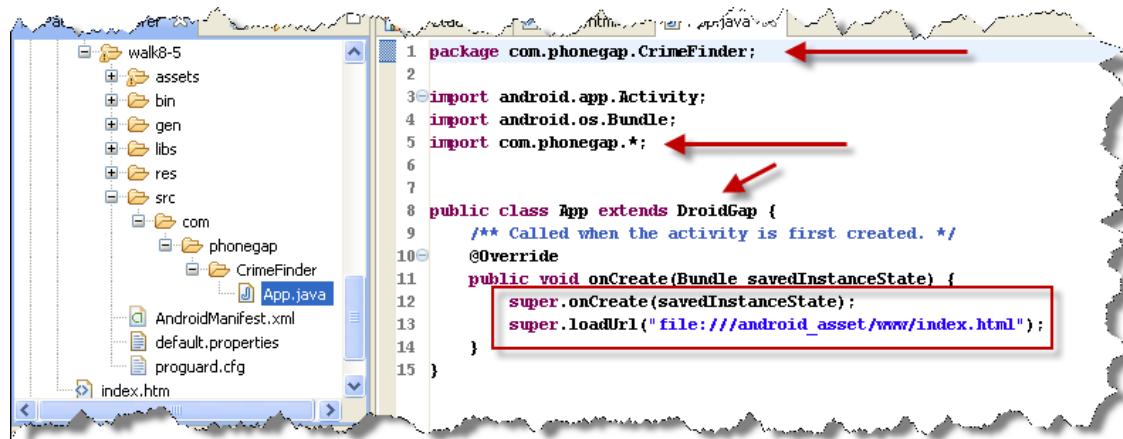
The ADT will add a number of directories and files to your Sencha Touch application. You will need to customize two of these files before you can compile and begin testing your native application.

You will need to make the following changes to the project's main Java file located in the src folder of your project:

1. Add `import com.phonegap.*;` to the collection of import statements
2. Change the class extend attribute from `Activity` to `DroidGap`
3. Replace the `setContentView()` line with

```
super.loadUrl("file:///android_asset/www/index.html");
```

After making the changes, your file should resemble the following screen shot:



## Tweaking AndroidManifest.xml

You will also need to modify your generated `AndroidManifest.xml` file, depicted below, by adding a series of directives after the `<manifest>` tag but before the `<application>` tag. You will also need to add the following attribute to the `<activity>` element:

```
    android:configChanges="orientation|keyboard|Hidden"
```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.phonegap.CrimeFinder"
    android:versionCode="1"
    android:versionName="1.0">

    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:resizeable="true"
        android:anyDensity="true"
    />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".App"
            android:label="@string/app_name" android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        
```

*Note: The bulk of the directives are provided to you in the snippets folder of your course files.*

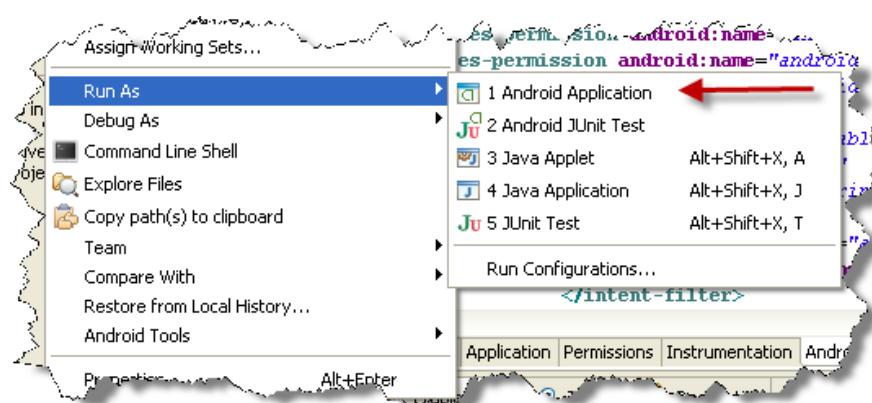
## Changing the Launch Icon

You can change the default launch icons for your application by editing the `icon.png` files in the following folders:

- /res/drawable-hdpi/icon.png
- /res/drawable-ldpi/icon.png
- /res/drawable-mdpi/icon.png

## Testing your Native Android Application

Once you have completed the aforementioned steps you can compile and launch your application in the Android simulator by right-clicking on your project in the Project Explorer and selecting Run As > Android Application as depicted below:



**Illustration 23: Launching your app in the emulator**

Note, however, that the performance of the Android simulator is very slow and may not accurately reflect a user's experience with their device. You should therefore install the app on as many devices as are available to you. In order to do this, simply copy your compiled application (the .apk file located in your project's /bin/ folder) to your handheld's SIM card. You can then install the application from the SIM using one of the many APK installers available on the Android Market including:

- Apk Manager
- Apk Installer
- Apps Installer

Another alternative to this approach is to install the app on a rooted Android device using Droid Explorer, located at the following URL:

<http://de.codeplex.com/releases/view/50997>

## Deploying your Application to the Android Market

Instructions for deploying your application to the Android Market are located at the following URL:

[http://wiki.phonegap.com/w/page/16494773/Getting-Ready-to-Deploy-to-Android-Market-\(Android\)](http://wiki.phonegap.com/w/page/16494773/Getting-Ready-to-Deploy-to-Android-Market-(Android))

## Demonstration 8-5: Compiling a Native Application



In this walkthrough, you will perform the following tasks:

- Create a new Android project
- Link to your Sencha Touch application
- Test your compiled application with the Android simulator

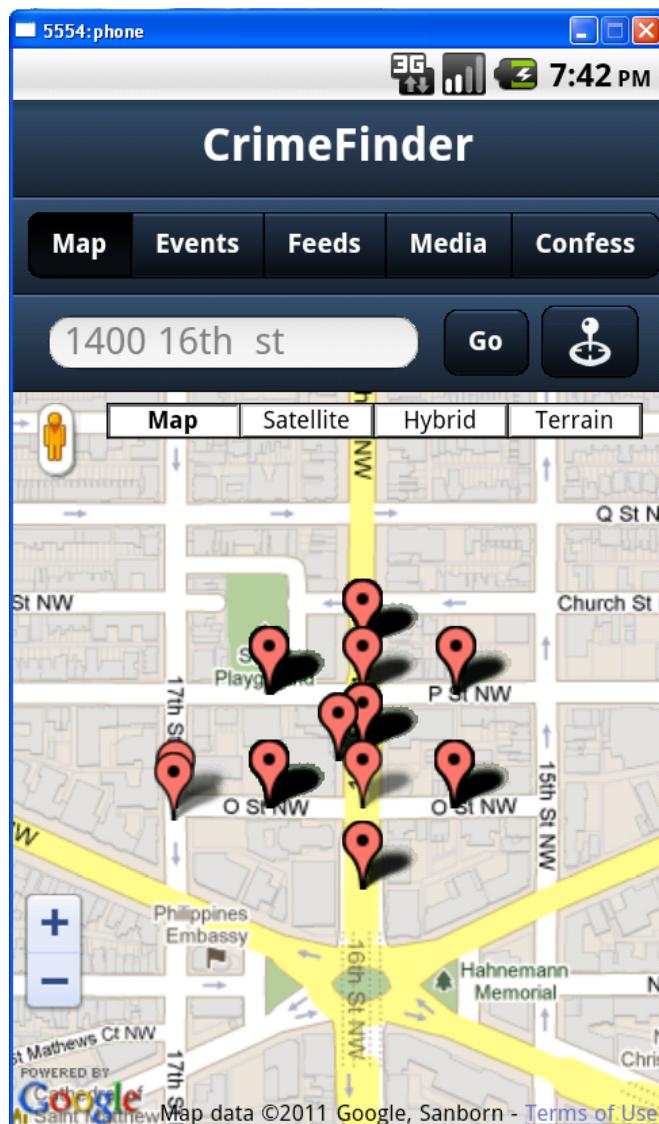


Illustration 24: CrimeFinder running as a native android app in the simulator

## Steps

### Create a new Android project

1. Download PhoneGap from [www.phonegap.com](http://www.phonegap.com)
2. In Aptana, select **File > New > Android Project**
3. Enter a project name of **CrimeFinder**
4. Change the default location to **/walk/walk8-5/**
5. Select a Build Target of **Android 2.3**
6. Enter the following properties:
  - Application name: **CrimeFinder**
  - Package Name: **com.phonegap.CrimeFinder**
  - Create Activity: **App**
7. Click **Finish**

### Add PhoneGap to the Project

8. In the **/walk/walk8-5/** folder, create a subdirectory named **libs**
9. Copy **/Phonegap/Android/phonegap.jar** file to **/walk8-5/libs**
10. Copy **/Phonegap/Android/phonegap.js** to **/walk8-5/assets/www**
11. Open the **CrimeFinder** project in Aptana
12. In the **assets/www/index.html** file, where indicated by the comment, insert a **<script>** tag that invokes the **phonegap.js** file. Your code should appear similar to the following:

```
<script
    type="text/javascript"
    charset="utf-8"
    src="phonegap-0.9.3.js"></script>
```

13. Wrap the **Ext.setup()** code block with a PhoneGap **deviceready** event listener as indicated by the following example:

```
document.addEventListener("deviceready", function() {
    Ext.setup({ // lots of code here })
})
```

14. Using Aptana, open **src/com/phonegap/CrimeFinder/App.java**

15. Add the following statement on line 5:

```
import com.phonegap.*;
```

16. Replace the **setContentView()** method with the following statement:

```
super.loadUrl("file:///android_asset/www/index.html");
```

17. Change the class's extend value from **Activity** to **DroidGap**

18. Save the file
19. Right-click on the /libs folder and select **Build Path > Configure Build Path**
20. Click on the **Libraries** tab
21. Click on the **Add JARs** button
22. Select the **CrimeFinder/libs/phonegap.jar** file
23. Click **OK**

#### **Configure the AndroidManifest**

24. Copy the contents of **/ftst/snippets/AndroidManifest.txt** to your clipboard
25. Open the **CrimeFinder/AndroidManifest.xml** file
26. Paste the contents of your clipboard directly above the **<application>** tag
27. Verify that the activity tag contains the following attribute:

```
android:configChanges="orientation|keyboardHidden"
```

#### **Test the Application**

28. In the Package Explorer view, right-click on the project and select Run As Android Application

## Unit Summary



- You can cache data locally using localstorage or Sqlite
- Your application code can be cached on the client by referencing a cache-manifest file
- Sencha MVC is a design pattern that enables teaming on projects and easier maintenance of your applications
- Sencha Command is a command-line utility that helps you quickly construct SMVC applications
- Your codebase should be compressed and minified before launch by using JSBuilder
- Sencha Touch applications may be converted into native applications using PhoneGap or QuickConnect

## Unit Review



1. What are the benefits of using LocalStorage?
2. What are the relative advantages/disadvantages of using Local Storage instead of local database?
3. How can you programmatically determine if your application has network access available?
4. What changes to your application would need to be made in order to fully support off-line execution?
5. Reconfiguration of your web server must occur in order to support application caching (true/false)?
6. What are the benefits of using Sencha MVC?
7. How can you test a Sencha-PhoneGap app on your Android device?

(This page intentionally left blank)

---

---

# **Appendix A: Installing the Courseware**

## **Unit Objectives**

After completing this unit, you should be able to:

- Install your development environment

## **Unit Topics**

- Basic Installation
- Installing Server Processes on Windows (optional)
- Installing Development Tools on Windows (optional)
- Deploying the Student Kit (optional)

## **Overview**

Fast Track to Sencha Touch has been designed to operate in two distinct environments. Following the basic setup instructions enables you to develop two applications that read data from static JSON files and a remote web service. Forgoing the Basic Installation in favor of the Optional installation processes enables you to pull dynamic data into your apps from live local web services that were developed using the ColdFusion Application Server and MySQL.

Following either set of setup instructions enables you to participate fully in the class.

## Walkthrough A-1: Basic Installation



In this walkthrough, you will perform the following tasks:

- Install the Apache HTTP Server
- Deploy the Course Files
- Install Aptana Studio

Note: You do not need to install a separate instance of Apache if you already have one installed (i.e. you are using XAMPP)

### Steps

#### **Install the Apache Web Server (if you do not already have an HTTP server installed)**

1. Download and install the stable release of the Apache Web Server (2.2.17) from <http://httpd.apache.org/download.cgi>. Accept all default settings EXCEPT the following:
  - Install Apache HTTP Server programs and shortcuts only for the Current User, on Port 8080, when started Manually
  - Set c:\Apache as the installation directory
2. Once installation of Apache is complete, using a text editor, open the file **c:\apache\conf\httpd.conf**
  3. Change the **Listen** port to **8100**
  4. Verify the **DocumentRoot** is **c:/apache/htdocs**
5. Run the Apache web server by double-clicking on c:\apache\bin\httpd.exe

#### **Install Safari**

6. Download and install Safari 5 from <http://www.apple.com/safari/download/>. Note that installation of the BonJour subcomponent is optional

#### **Install Aptana Studio**

7. Download the standalone version of Aptana Studio 2 from the following url: [www.aptana.com](http://www.aptana.com). **Do NOT install Aptana Studio 3**
8. Install Aptana Studio 2, accepting the default option

### **Deploy Sencha Touch and the Course Files**

9. Download Sencha Touch from the following URL:  
<http://www.sencha.com/products/touch/>
10. Unzip **Sencha Touch** to **/apache/htdocs/**
11. Rename the Sencha Touch directory to **senchatouch**
12. Download Sencha Touch Charts from the following URL:  
<http://www.sencha.com/products/charts/download/>
13. Unzip the file into **/apache/htdocs/**
14. Download the course files from the following url:  
<http://www.senchatraining.com/ftst.zip>
15. Unzip the course files to **/apache/htdocs/**. This will create an **/apache/htdocs/ftst/** folder.

**-- End of Walkthrough --**

## Walkthrough A-2: Installing Server Processes on Windows (optional)



In this walkthrough, you will perform the following tasks:

- Install the Apache HTTP Server
- Install the ColdFusion Server
- Install MySQL

### Steps

#### Install the Apache Web Server

1. Download and install the stable release of the Apache Web Server (2.2.17) from <http://httpd.apache.org/download.cgi>. Accept all default settings EXCEPT the following:
  - Install Apache HTTP Server programs and shortcuts only for the Current User, on Port 8080, when started Manually
  - Set **c:\Apache** as the installation directory
2. Once installation of Apache is complete, using a text editor, open the file **c:\apache\conf\httpd.conf**
  - Change the **Listen** port to **8100**
  - Verify the **DocumentRoot** is **c:/apache/htdocs**
3. Run the Apache web server by double-clicking on **c:\apache\bin\httpd.exe**

#### Install Coldfusion 9

4. Download and install the Adobe ColdFusion 9 trial from <http://www.adobe.com/cfusion/tcrc/index.cfm?product=coldfusion>
5. Accept all of the default settings except the following:
  - When prompted for a serial number, turn on the checkbox for 30-day trial
  - On the subcomponents screen, deselect all the options
  - On the Configure Web Servers/Websites screen, click the Add button specify the following:

Web Server: **Apache**

Configuration Directory: **c:\apache\conf**

Directory and file name of server binary: **c:\apache\bin\httpd.exe**

- When prompted for passwords, use the word **password**

6. After ColdFusion 9 has finished installing, restart Apache
7. Open a web browser and go to  
**<http://localhost:8100/CFIDE/administrator/index.cfm>**
8. Follow the on-screen prompts. When the ColdFusion Administrator appears, bookmark the URL in the browser.

#### **Install MySQL Community Server**

9. Download MySQL Community Server from  
**<http://dev.mysql.com/downloads/mysql/>**
10. Install MySQL Community Server, accepting all defaults except the following:
  - When prompted to select a configuration type, select **Standard Configuration**
  - Enter a root password of **password**

#### **Install MySQL Workbench**

11. Download mySQLWorkbench from  
**<http://dev.mysql.com/downloads/workbench/>**
12. Install **mySQL Workbench** using the default selections.
13. Run **mySQL Workbench**
14. In the **Server Administration** column, click on **New Server Instance**
15. Click on the **Store in Vault** button and enter the root password –  
**password**
16. Click **Next**
17. Click **Next**
18. Click **Next**
19. Click **Finish**

**– End of Walkthrough --**

## Walkthrough A-3: Installing Development Tools on Windows (optional)



In this walkthrough, you will perform the following tasks:

- Install Safari
- Install Aptana Studio 2
- Install the Android SDK

### Steps

#### Install Safari

1. Download and install Safari 5 from <http://www.apple.com/safari/download/>. Note that installation of the BonJour subcomponent is optional.

#### Install Aptana Studio

2. Download the standalone version of Aptana Studio 2 from the following url: [www.aptana.com](http://www.aptana.com)
3. Install Aptana Studio 2, accepting the default options

#### Install the Android SDK

4. Download the Android SDK from <http://developer.android.com/sdk/index.html>
5. Unzip the Android SDK to C:\
6. Right-click on **My Computer** and select **Properties**
7. Click on the **Advanced tab** and then click on **Environment Variables**
8. In the System Variables section, click on **Path** and click **Edit**
9. Add the following location to the path variable:

**C:\android-sdk-windows\tools**

10. Click **OK**
11. Click **OK**
12. Run **C:\android-sdk-windows\tools\android.bat**
13. Click on **Available Packages**
14. Turn on the checkbox labeled **SDK Platform Android 2.2, API 8, revision 2**

15. Click **Install Selected**
16. Click **Install**
17. Once the package has completed downloading, click **Close**

**– End of Walkthrough --**

## Walkthrough A-4: Deploying the Student Kit (optional)



In this walkthrough, you will perform the following tasks:

- Unzip Sencha Touch
- Unzip the course files
- Restore the MySQL Database
- Define a ColdFusion Datasource

### Deploy Sencha Touch and the Course Files

1. Unzip **Sencha Touch** to **/apache/htdocs/**
2. Rename the Sencha Touch directory to **senchatouch**
3. Unzip the course files to **/apache/htdocs**. This will create an **/apache/htdocs/ftst/** folder.

### Restore the Database

4. Open MySQLWorkbench
5. Under the Server Administration column, click on New Server Instance
6. Click the **Continue** button for all of the dialog boxes until you reach the **Finish** button
7. Click on the **Finish** button
8. Click on **Manage Import / Export**
9. Click on [mysqld@localhost](#) and click OK
10. Click on the **Data Dump** tab
11. Click on the **Import from Disk** tab
12. Click the radio button labeled **Import from Self-Contained File**
13. Select the following file to import:

**/apache/htdocs/ftst/database/ftst.sql**

14. Click **Start Import**
15. Once the import has completed close MySQL Workbench

### Configure a ColdFusion Datasource

16. Open a web browser to the following url:

**<http://localhost/CFIDE/administrator/index.cfm>**

17. Enter your password and click Login
18. Under the Data & Services heading, click Data Sources
19. Enter a Data Source Name of ftst
20. Select **MySQL (4/5)** as the database driver
21. Click the **Add** button
22. Enter the following information:
  - Database: **ftst**
  - Server: **127.0.0.1**
  - Port: **3306**
  - Username: **root**
  - Password: **[your password]**
23. Click **Submit**

**-- End of Walkthrough --**