ICT286
Web and Mobile Computing

Topic 6
Cross-Platform Mobile Apps Development with Cordova

Objectives

- Understand the pros and cons of cross-platform mobile apps written with Cordova and native apps written with platform-specific languages such as Android and iOS.
- For a mobile app written with Cordova, understand the relationship between HTML/CSS/JavaScript code written by the application programmer and the Cordova library, Cordova plugins it uses, and the native code that implements those Cordova plugins.
- Understand the directory structure of a Cordova project, in particular, the purpose of the following directories in a Cordova project: `www` directory, the `platforms` directory and the `plugins` directory.

Objectives

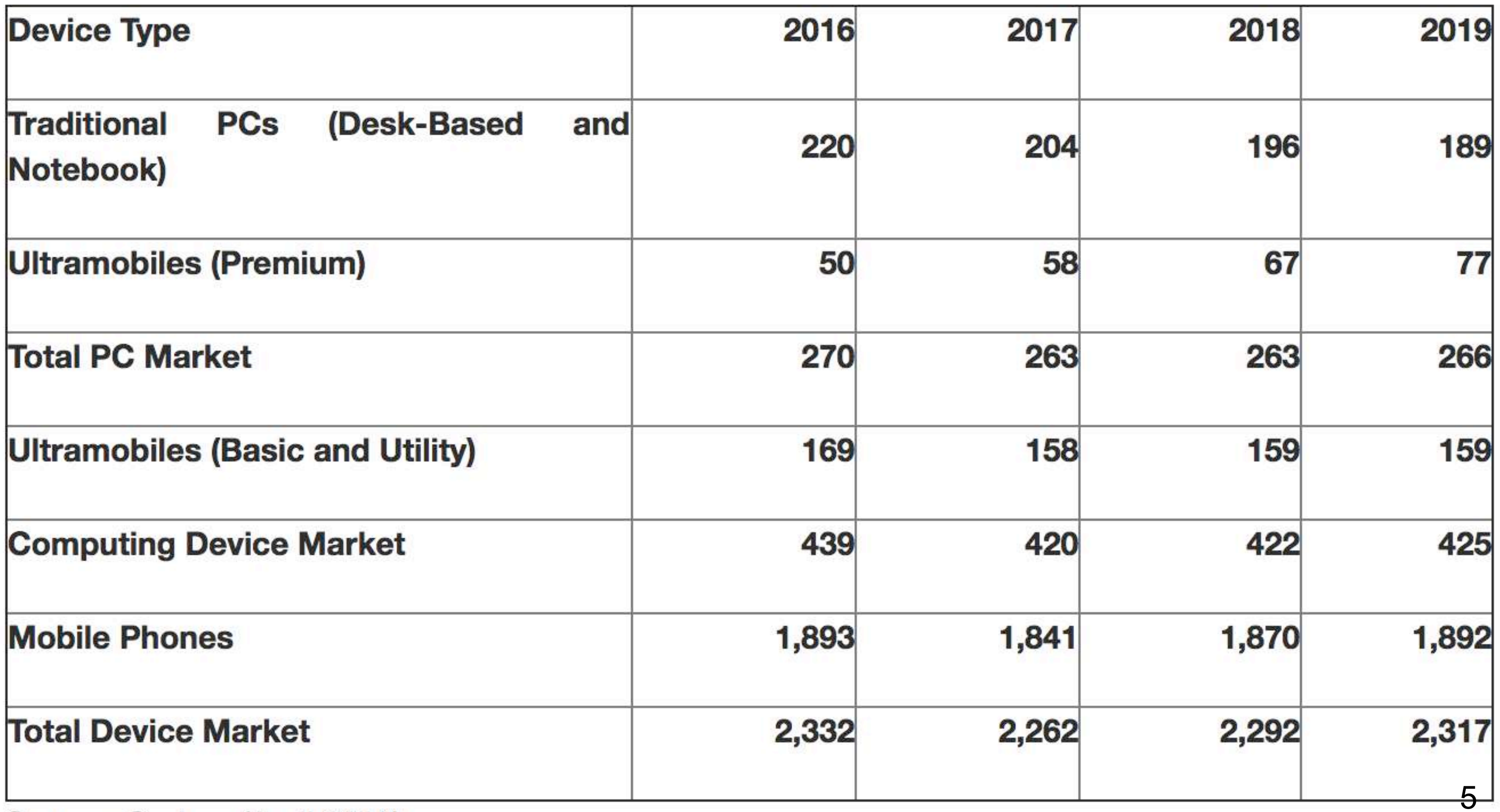
- Be able to install Cordova and related programs on your computer so that you can develop Cordova apps on your own computer.
- Be able to use Cordova commands to package apps for multiple platforms including web browser, Android and iOS.
- Be able to build and run apps on web browser, Android emulator and iOS emulator.
- Understand and be able to design simple cross-platform mobile apps using Cordova.
- Be able to access device resources including device information, accelerometer and camera.

Smart Phones

- Blackberry smart phone appeared in 2003 – the first widely used mobile phone for accessing the Web
- Unlike the old mobile phones, the smart phones are controlled by sophisticated operating systems
- Based on the OS, smart phone market is now dominated by Google’s Android, and Apple’s iOS.
- Tablets usually use the same operating systems as their smart phone counterpart.

Smart Phones vs PCs: Annual Shipment

Worldwide Device Shipments by Device Type, 2016-2019 (Millions of Units)



Device Type	2016	2017	2018	2019
Traditional PCs (Desk-Based and Notebook)	220	204	196	189
Ultramobiles (Premium)	50	58	67	77
Total PC Market	270	263	263	266
Ultramobiles (Basic and Utility)	169	158	159	159
Computing Device Market	439	420	422	425
Mobile Phones	1,893	1,841	1,870	1,892
Total Device Market	2,332	2,262	2,292	2,317

Source: Gartner (April 2018)

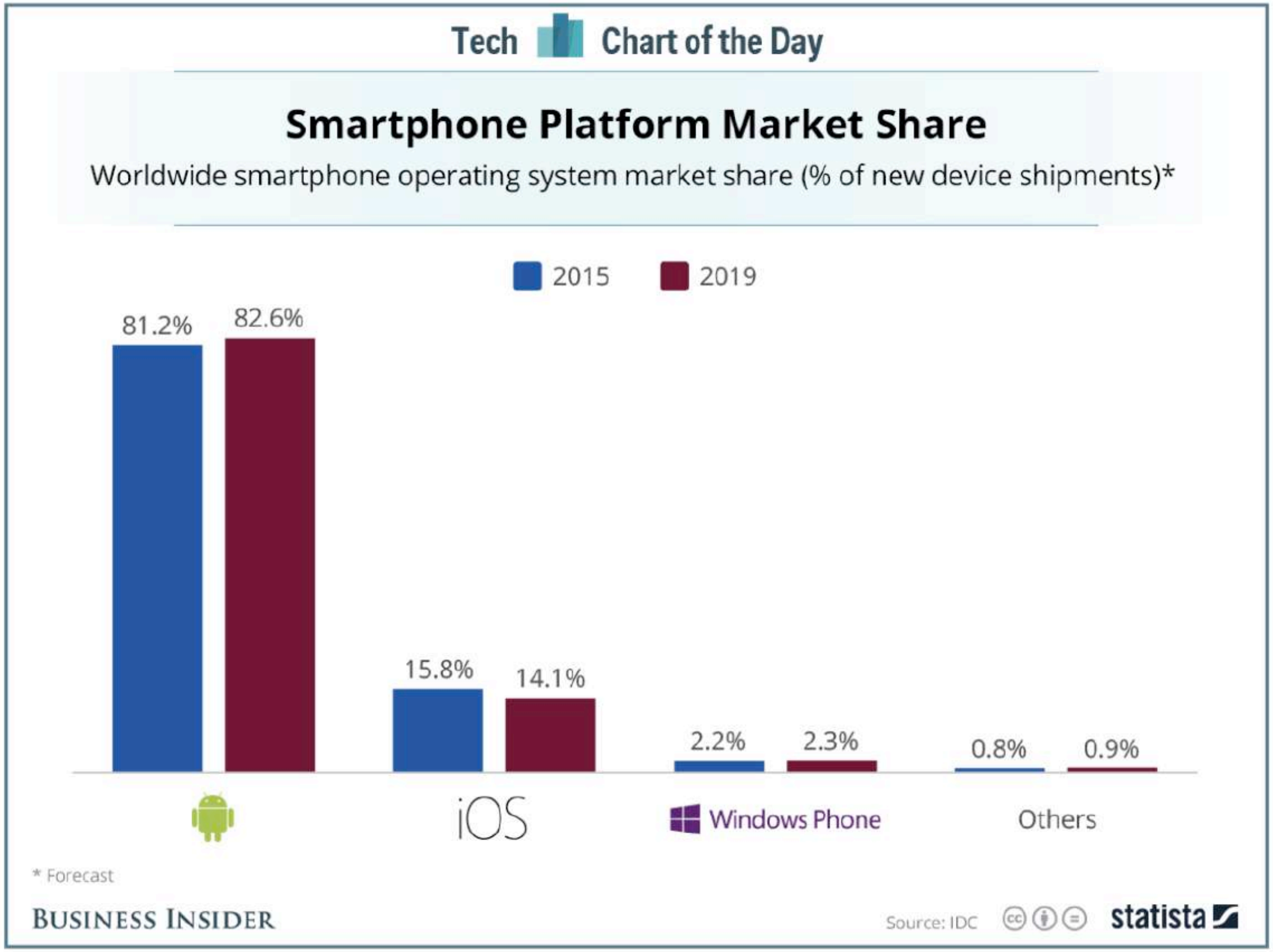
Smart Phones vs PCs: Functionality

- PCs generally used the wired network or WiFi to access the Internet, their mobility is limited.
- Smart phones use two communications methods: WiFi and cell phone networks, hence they are much more mobile.
- Compared to conventional PCs, smart phones are equipped with
  - Touch screen, hence with much natural and user-friendly interface
  - GPS and/or other global positioning systems, therefore they are capable of being location-aware
  - Accelerometers, enabling the detection of device rotation and motion and various other sensors.

Smart Phones vs PCs: Limitations

- PCs generally have large screen size than those on the smart phones. This means some applications need to be adapted to work on the small screens, sometimes requiring a major re-design of their screen layout.
- The CPU and GPU on the smart phones generally run at slower speeds compared to those on the PCs due to the limitation of physical size, power consumption, and heat problem, hence in many cases, the CPU intensive operations need to be shifted to the server end and the mobile app acts just as the front end of the application.
- Smart phones are run by small batteries and this is a serious limitation compared to desktop and even laptop computers.

Market Share of Smart Phone OS



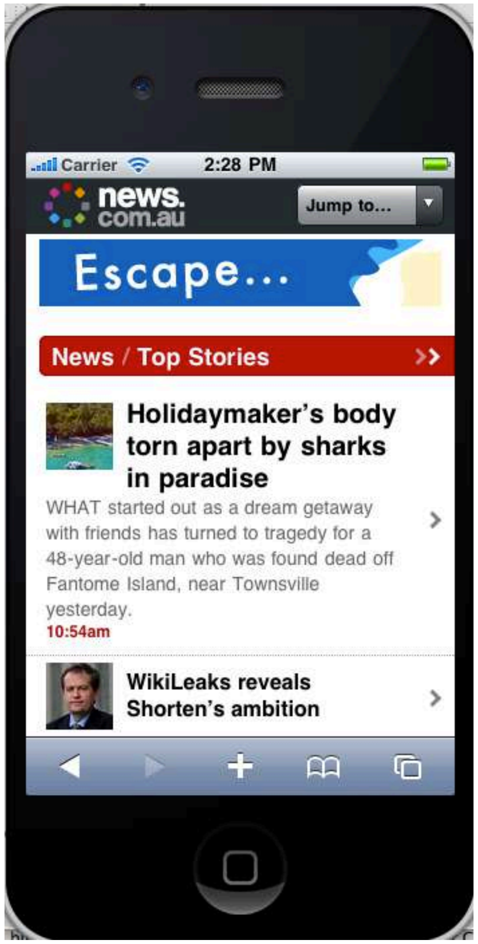
Android and iOS Eco-Systems

- Both Apple’s iOS and Google’s Android have developed their own app eco-systems, including their online app stores
- According to http://www.statista.com, by 2nd quarter 2019:
  - Andoid Play Store: 2.46 millions apps
  - Apple’s App Store: 1.96 millions apps
  - Windows Store: 0.67 millions apps
- This new way of software development, marketing and sales reaches out to millions of potential app developers worldwide

Mobile Web

- a website that is specifically optimized for the mobile devices
- the user interface is built with web-standard technologies, such as HTML, CSS and JavaScript
- available at a URL, the contents are all stored on the web server.
- a mobile web is not installed on the phone. Its contents are retrieved from its server and rendered inside a web browser running on the mobile device
- there is usually a full version for PCs and the difference between the full version and mobile version is mainly in user interface, not in their functionalities.

Mobile Web


Carrier 2:28 PM

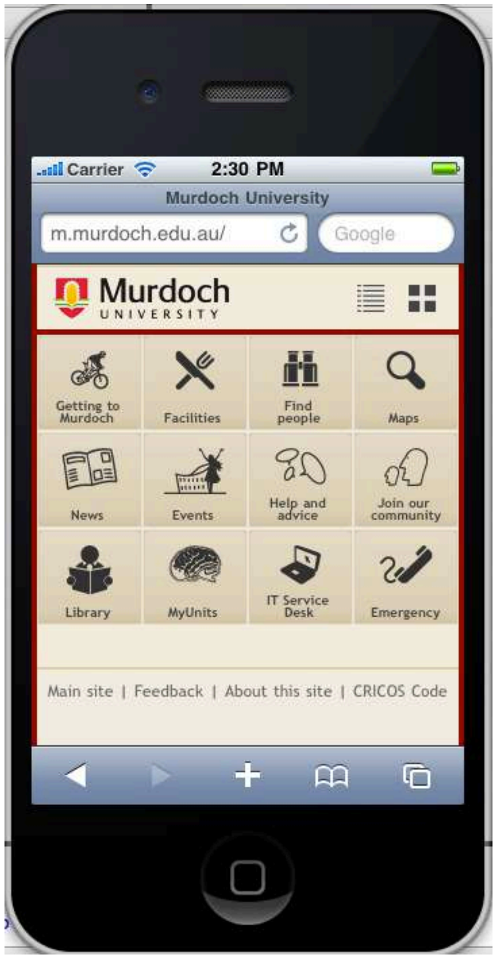
news.com.au Jump to...

Escape...

News / Top Stories

Holidaymaker's body torn apart by sharks in paradise
WHAT started out as a dream getaway with friends has turned to tragedy for a 48-year-old man who was found dead off Fantome Island, near Townsville yesterday.
10:54am

WikiLeaks reveals Shorten's ambition


Carrier 2:30 PM
Murdoch University
m.murdoch.edu.au/ Google

Murdoch UNIVERSITY

Main site | Feedback | About this site | CRICOS Code

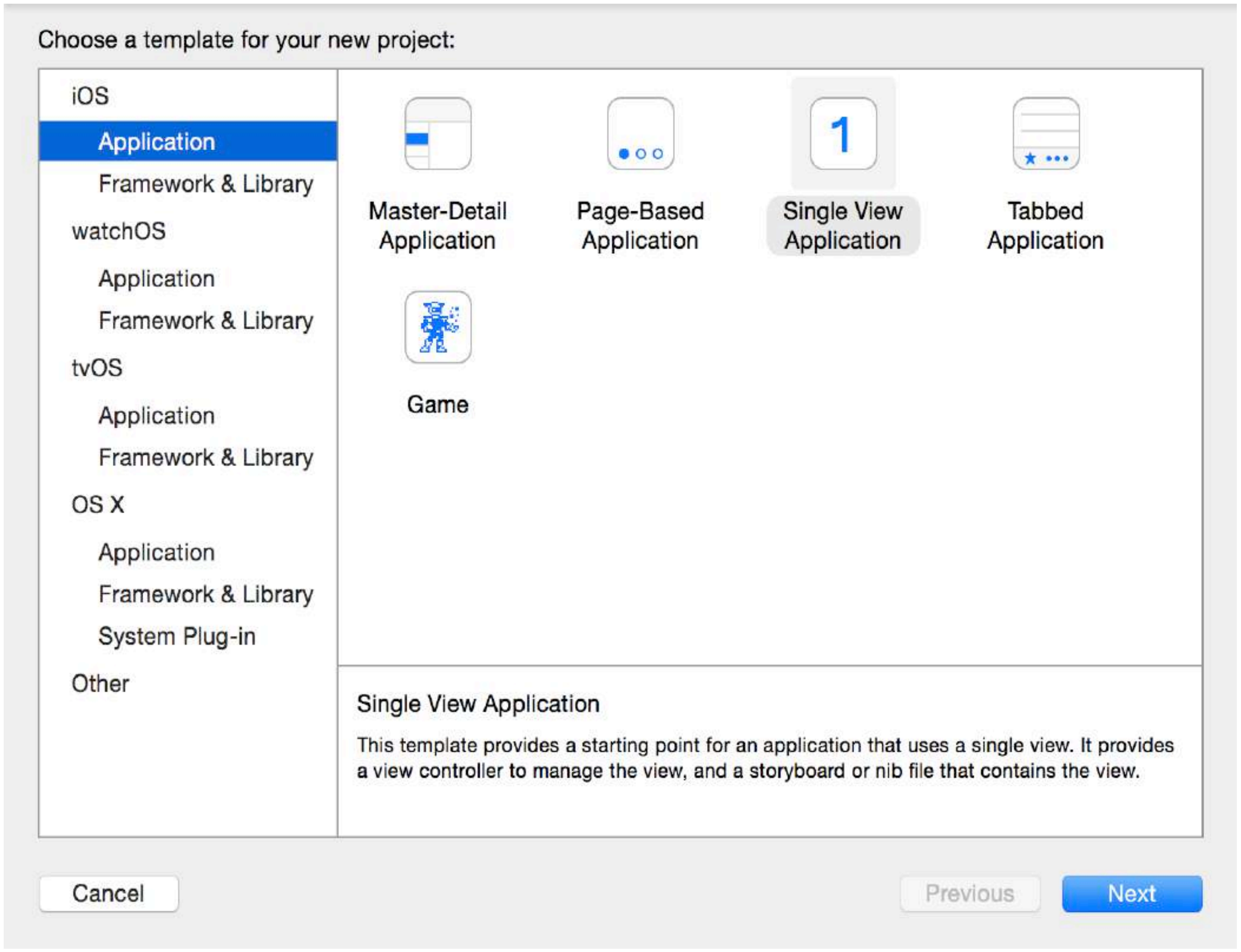
Mobile App

- is an application program designed for a specific task
- most of its contents are installed on the mobile device and the app run directly on the device (rather than inside a web browser)
- it has access to the device hardware (speakers, GPS, accelerometer, camera, file system, etc.)
- written with a programming language, such as Swift for iOS, and Java for Android
- there are also development libraries and tools such as Apache Cordova that allow development of cross-platform mobile apps. In this case, the programmer writes one app, but the app can be built for multiple platforms such as Android, iOS, Windows, macOS, Linux etc.

Develop Mobile Apps - iOS

- Require Apple Xcode to develop native iOS applications.
- Xcode is only available on macOS, therefore you cannot develop a native iOS app using Windows or Linux
- Applications are written with a new programming language Swift
- Need to register as an Apple Developer
- The app needs to be distributed via Apple’s App Store.
- The app can only run on the iOS platform

Xcode and Swift

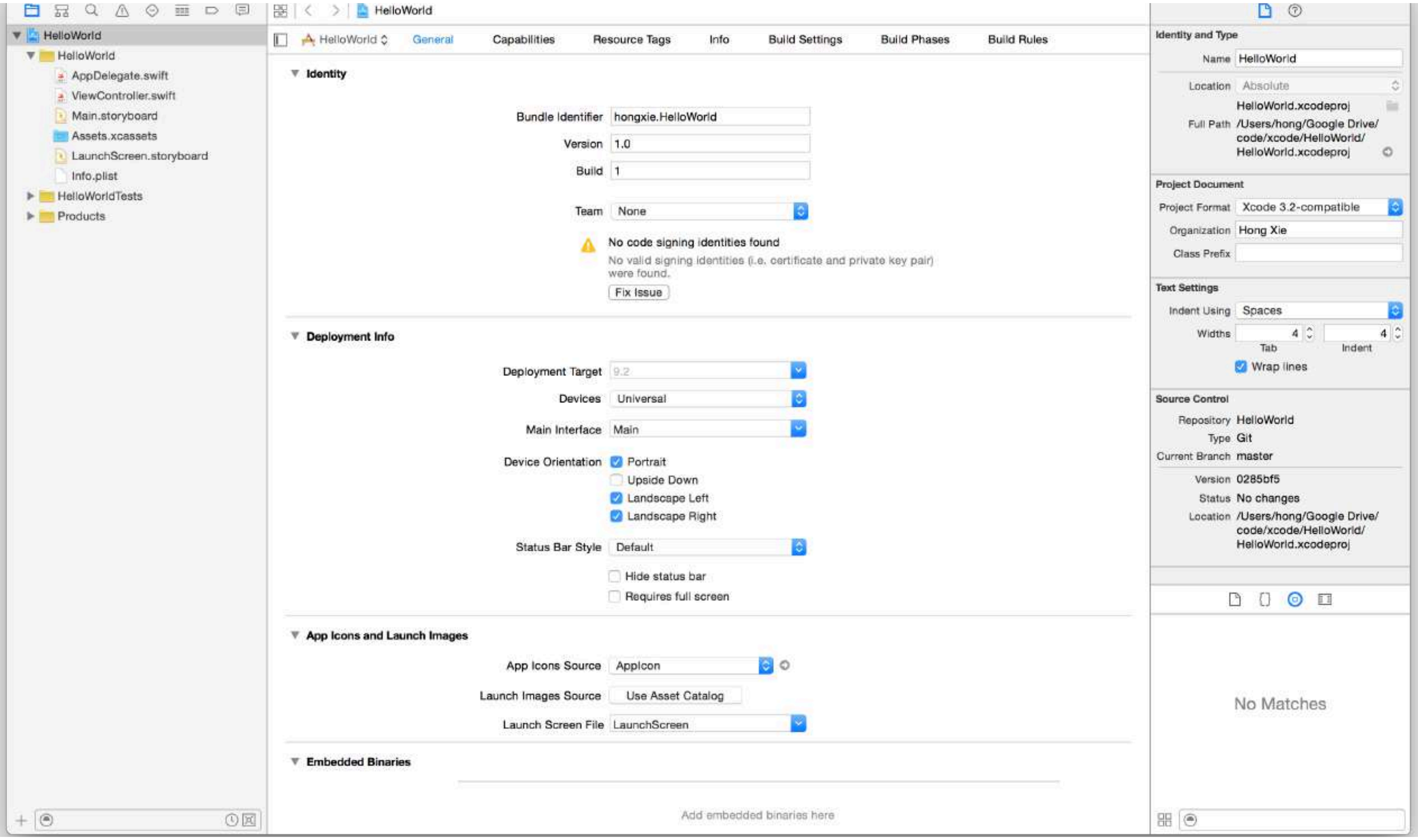


Choose a template for your new project:

Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Xcode and Swift



Identity

Bundle Identifier: hongxie.HelloWorld
Version: 1.0
Build: 1
Team: None

No code signing identities found
No valid signing identities (i.e. certificate and private key pair) were found.
Fix Issue

Deployment Info

Deployment Target: 9.2
Devices: Universal
Main Interface: Main
Device Orientation: ✓ Portrait, ☐ Upside Down, ✓ Landscape Left, ✓ Landscape Right
Status Bar Style: Default
☐ Hide status bar
☐ Requires full screen

App Icons and Launch Images

App Icons Source: AppIcon
Launch Images Source: Use Asset Catalog
Launch Screen File: LaunchScreen

Embedded Binaries

Add embedded binaries here

Identity and Type

Name: HelloWorld
Location: Absolute
HelloWorld.xcodeproj
Full Path: /Users/hong/Google Drive/code/xcode/HelloWorld/HelloWorld.xcodeproj

Project Document

Project Format: Xcode 3.2-compatible
Organization: Hong Xie
Class Prefix:

Text Settings

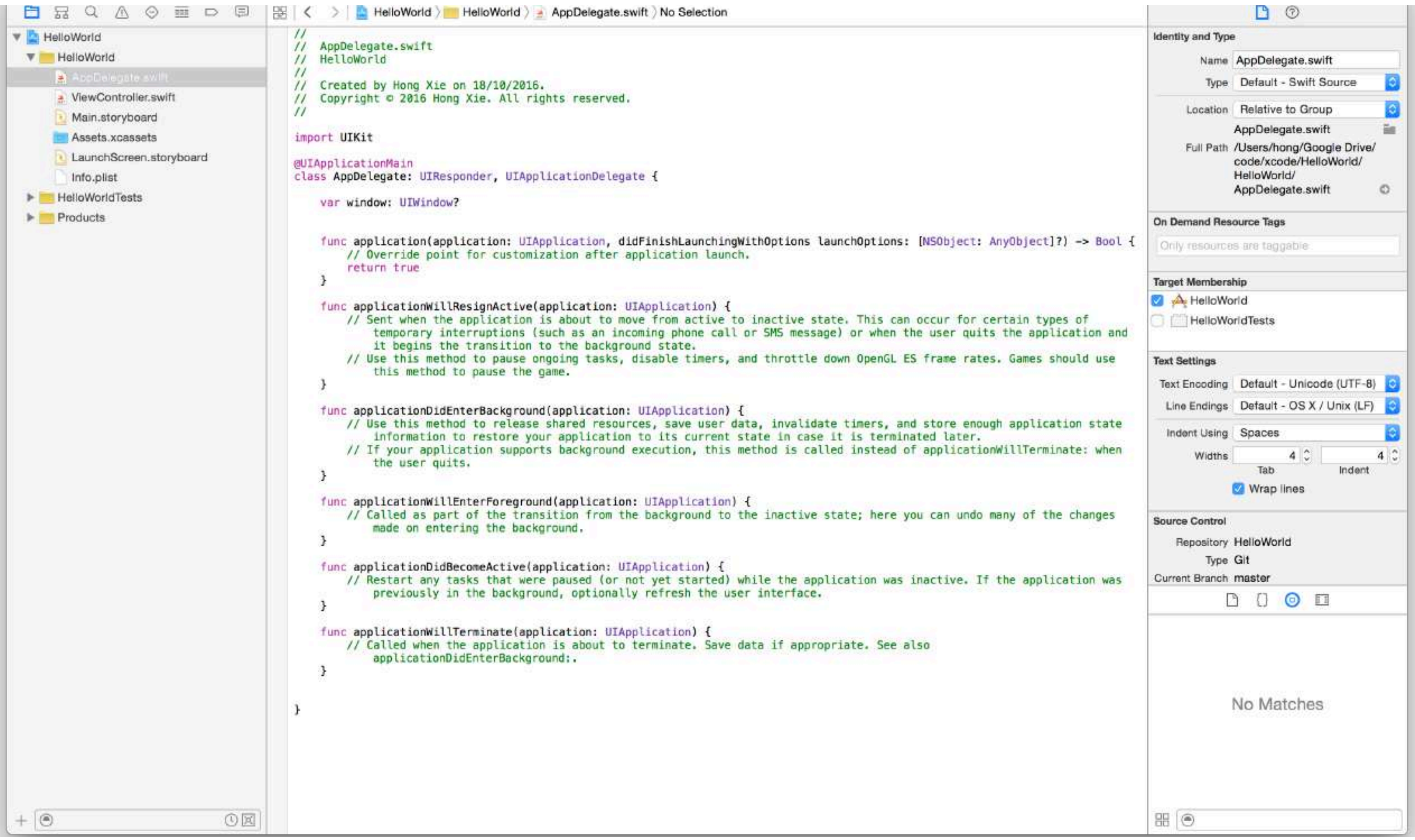
Indent Using: Spaces
Widths: 4 (Tab) 4 (Indent)
✓ Wrap lines

Source Control

Repository: HelloWorld
Type: Git
Current Branch: master
Version: 0285bf5
Status: No changes
Location: /Users/hong/Google Drive/code/xcode/HelloWorld/HelloWorld.xcodeproj

No Matches

Xcode and Swift



```swift
//
//  AppDelegate.swift
//  HelloWorld
//
//  Created by Hong Xie on 18/10/2016.
//  Copyright © 2016 Hong Xie. All rights reserved.
//

import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for certain types of
            temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and
            it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use
            this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough application state
            information to restore your application to its current state in case it is terminated later.
        // If your application supports background execution, this method is called instead of applicationWillTerminate: when
            the user quits.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you can undo many of the changes
            made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was
            previously in the background, optionally refresh the user interface.
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See also
            applicationDidEnterBackground:.
    }


}
```

Identity and Type

Name: AppDelegate.swift
Type: Default - Swift Source
Location: Relative to Group
AppDelegate.swift
Full Path: /Users/hong/Google Drive/code/xcode/HelloWorld/HelloWorld/AppDelegate.swift

On Demand Resource Tags

Only resources are taggable

Target Membership

✓ HelloWorld
☐ HelloWorldTests

Text Settings

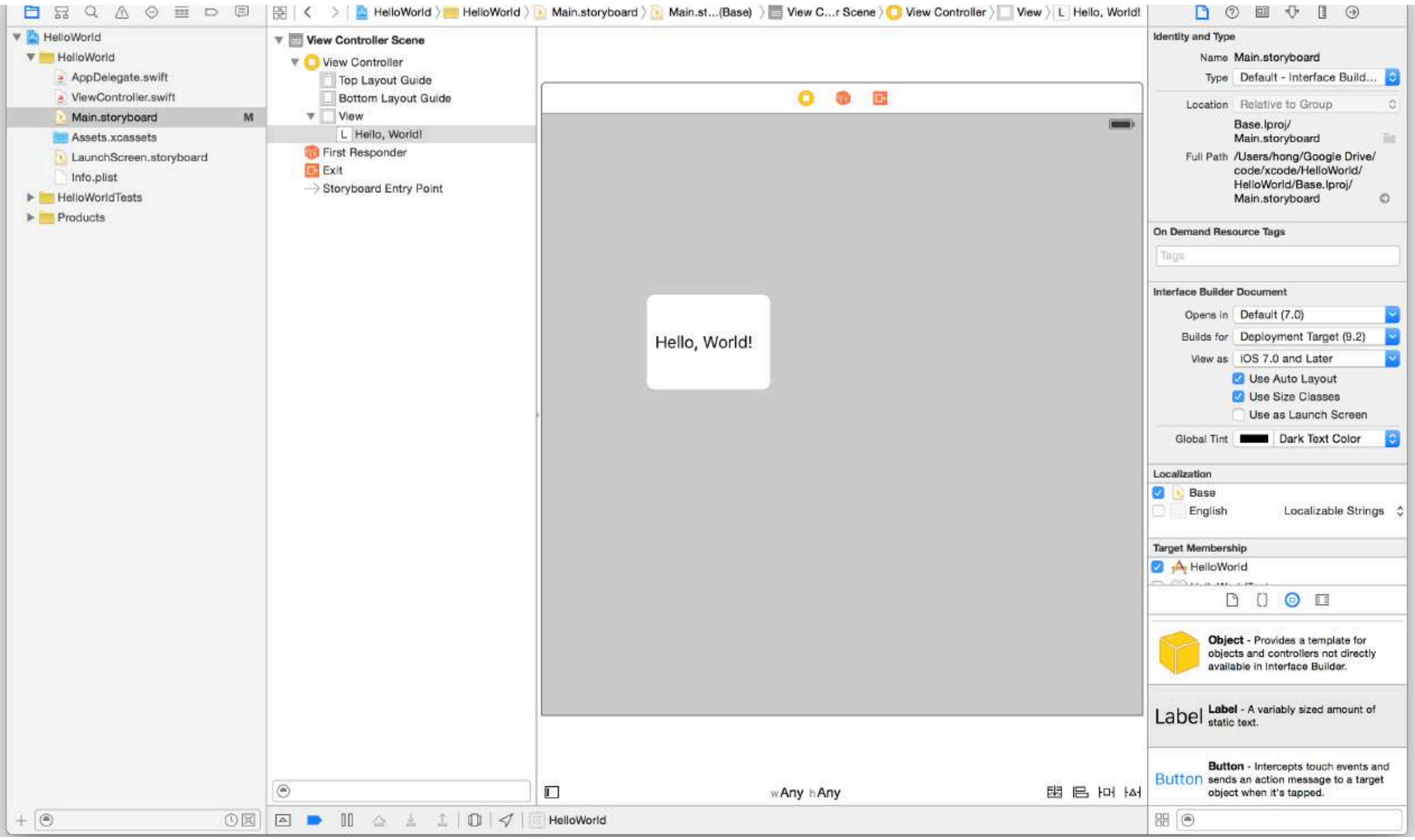
Text Encoding: Default - Unicode (UTF-8)
Line Endings: Default - OS X / Unix (LF)
Indent Using: Spaces
Widths: 4 Tab, 4 Indent
✓ Wrap lines

Source Control

Repository: HelloWorld
Type: Git
Current Branch: master

No Matches

Xcode and Swift



Identity and Type

Name Main.storyboard

Type Default - Interface Build...

Location Relative to Group

Base.lproj/Main.storyboard

Full Path /Users/hong/Google Drive/code/xcode/HelloWorld/HelloWorld/Base.lproj/Main.storyboard

On Demand Resource Tags

Tags

Interface Builder Document

Opens in Default (7.0)

Builds for Deployment Target (9.2)

View as iOS 7.0 and Later

☑ Use Auto Layout

☑ Use Size Classes

☐ Use as Launch Screen

Global Tint ■ Dark Text Color

Localization

☑ Base

☐ English Localizable Strings

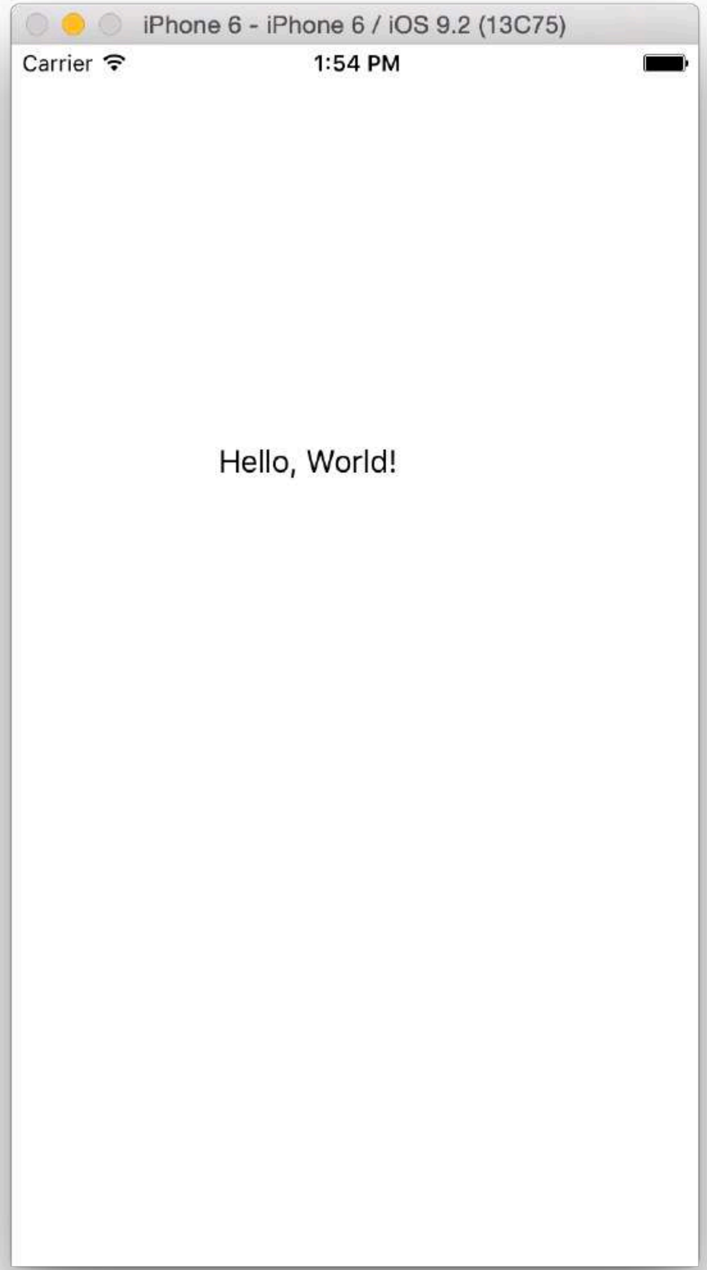
Target Membership

☑ HelloWorld

Object - Provides a template for objects and controllers not directly available in Interface Builder.

Label **Label** - A variably sized amount of static text.

Button **Button** - Intercepts touch events and sends an action message to a target object when it's tapped.



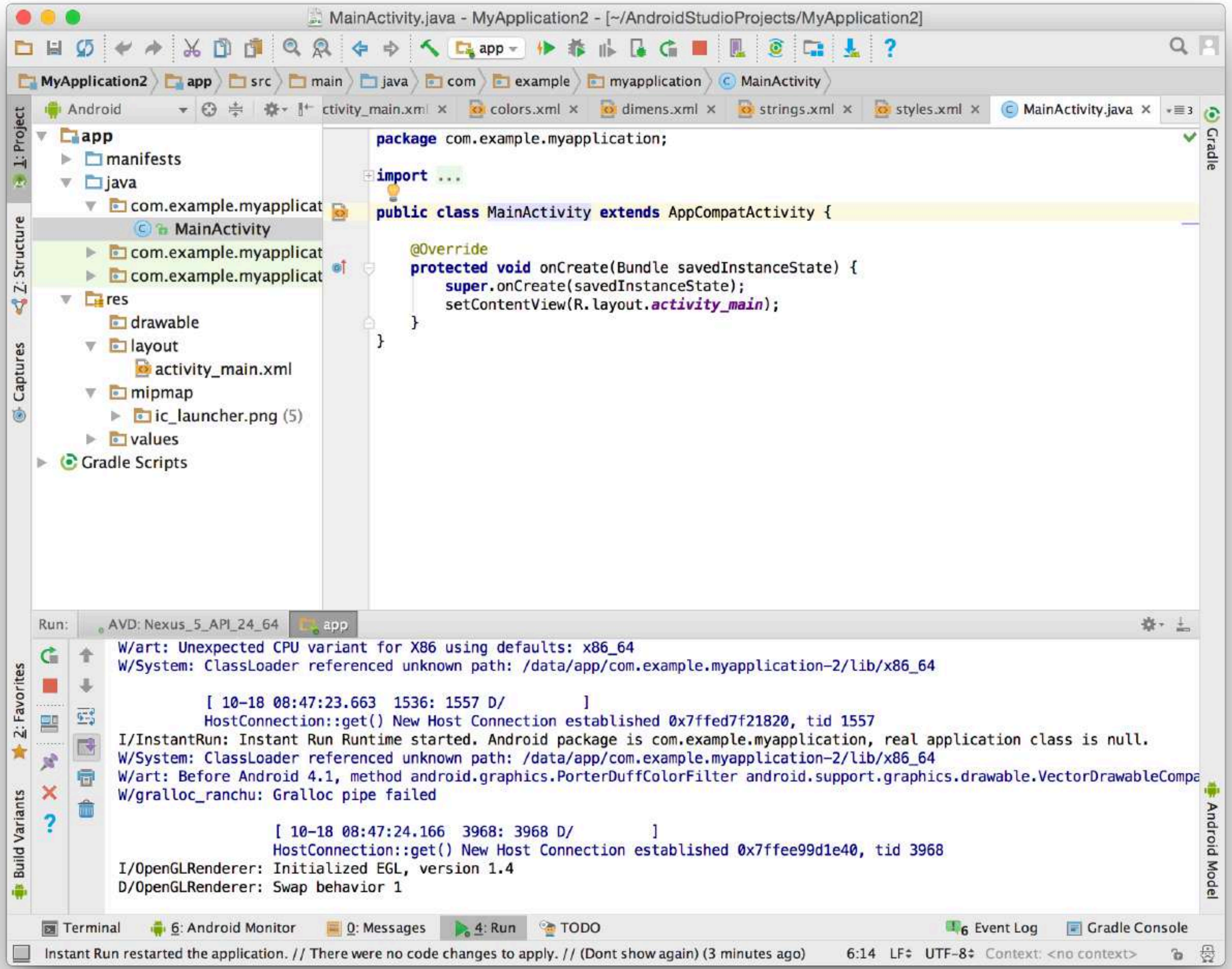
Hello, World!

Xcode and Swift

Develop Mobile Apps - Android

- Require Android Studio and its SDK to develop native Android applications.
- Android Studio and SDK are available on Windows, macOS, and Linux
- Android applications are written with Java programming language and XML
- An native Android application can only run on a device running Android operating system
- Android operating system uses Linux kernel
- Android has the largest installed base of any operating system

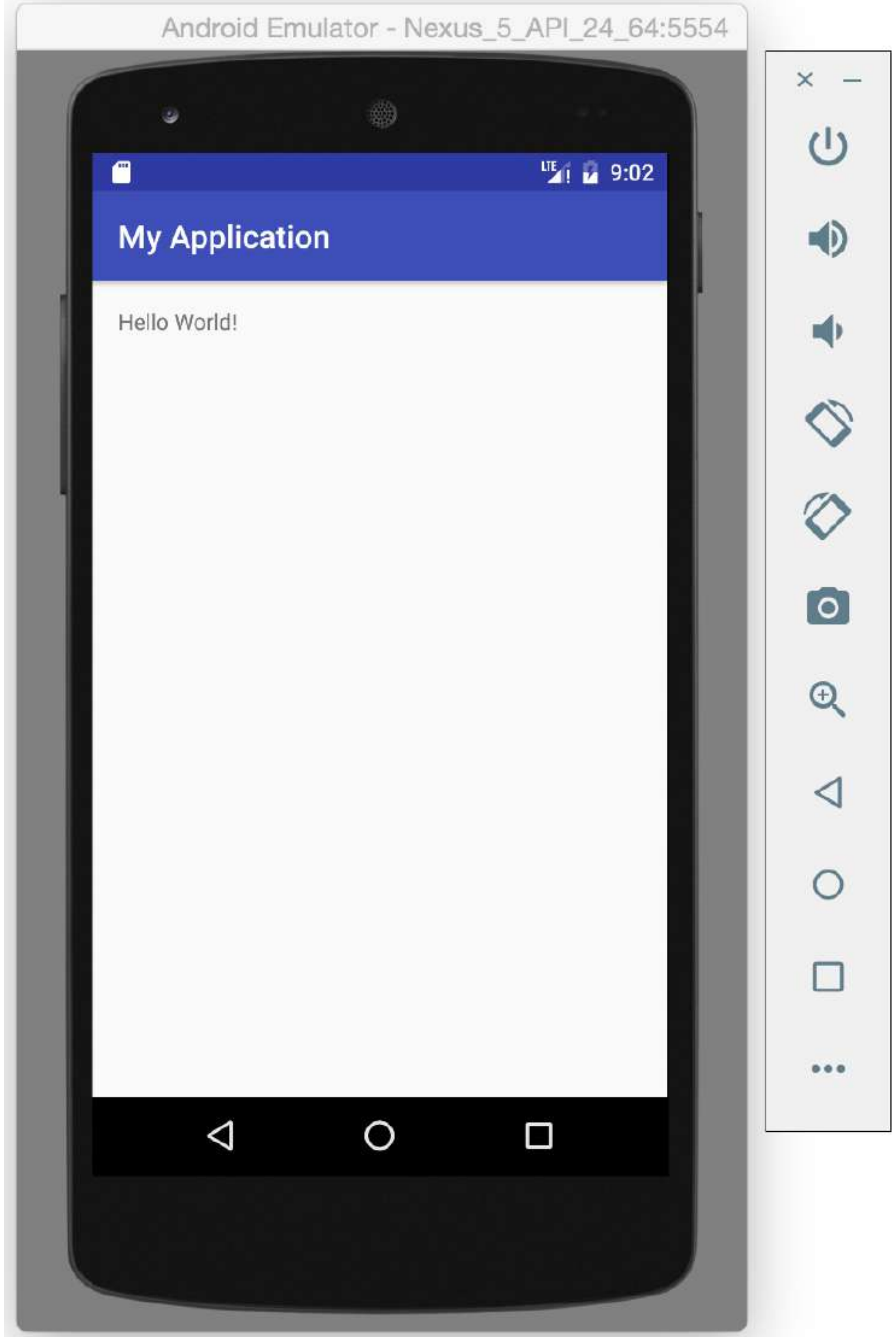
Android Studio and Java



```java
package com.example.myapplication;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

```
W/art: Unexpected CPU variant for X86 using defaults: x86_64
W/System: ClassLoader referenced unknown path: /data/app/com.example.myapplication-2/lib/x86_64

           [ 10-18 08:47:23.663  1536: 1557 D/         ]
           HostConnection::get() New Host Connection established 0x7ffed7f21820, tid 1557
I/InstantRun: Instant Run Runtime started. Android package is com.example.myapplication, real application class is null.
W/System: ClassLoader referenced unknown path: /data/app/com.example.myapplication-2/lib/x86_64
W/art: Before Android 4.1, method android.graphics.PorterDuffColorFilter android.support.graphics.drawable.VectorDrawableCompa
W/gralloc_ranchu: Gralloc pipe failed

                   [ 10-18 08:47:24.166  3968: 3968 D/         ]
                   HostConnection::get() New Host Connection established 0x7ffee99d1e40, tid 3968
I/OpenGLRenderer: Initialized EGL, version 1.4
D/OpenGLRenderer: Swap behavior 1
```

Instant Run restarted the application. // There were no code changes to apply. // (Dont show again) (3 minutes ago)



Android Studio and Java

Cross-Platform Mobile Apps

- iPhone mobile apps developed with Xcode only runs on iPhone and iPad
- Need to learn a new language, Objective-C or Swift
- Android apps developed with Android Studio only runs on Android devices
- Need to use Java
- This means one has to develop the same app at least twice, one for each major platform

Cross-Platform Mobile Application Development

- An alternative to these platform-specific development environments, such as Xcode and Android Studio, is Apache Cordova.
- Cordova:
  - open-source mobile apps development framework
  - use standard web technologies you are already familiar with: HTML5, CSS, and JavaScript
  - provide a consistent JavaScript API to access the underlying hardware and system features
  - available on Windows, Mac and Linux
  - work with a consistent command line interface (CLI) on all development platforms

Apache Cordova

- A Cordova application is written in HTML5, CSS and JavaScript
- The application is rendered by the same web engine that the mobile device’s web browser uses to render web pages.
- However, the standard web application is not able to access and manipulate the mobile device’s hardware such as camera, battery information, accelerometer, and many system features such as contacts and file system.
- For this the Cordova team developed a set of JavaScript libraries that provide one consistent API for accessing these hardware and system features cross several different mobile OS platforms.

Cordova Source Code

- The source code of a Cordova application consists of two parts:
  - Application program code, written by the application programmer. The program is coded in HTML5, CSS and JavaScript. The JavaScript code includes calls to Cordova libraries to access hardware and system features.
  - Cordova code, developed by the Cordova team. The code consists of JavaScript code as well as platform-specific source code that implements the Cordova API for a specific platform, using platform-specific language such as Swift for iOS, and Java for Android.

Cordova Plugins

- Cordova library is organised as a set of “plugins”
- Each plugin provides JavaScript API for a set of hardware and system features, eg
  - **cordova-plugin-camera** defines a global `navigator.camera` object, that provides the API for accessing the cameras on the device, eg:
    - `.getPicture(successCallback, errorCallback, options)`
    - `.cleanup()`
    - `.CameraOptions : Object`
  - **cordova-plugin-device-motion** provides access to the device's accelerometer via `navigator.accelerometer` object.
  - **cordova-plugin-contacts** defines a global `navigator.contacts` object, which provides access to the device contacts database.

Cordova Plugins

- For each OS platform, there is a separate implementation of Cordova plugins for that OS platform, eg:
  - There is an implementation of Cordova plugins for iOS platform.
  - There is an implementation of the same plugins for Android platform.
  - There is an implementation of the same plugins for Windows Phone platform.
- For each supported platform, a Cordova plugin is implemented with the language specific to that platform using the API provided by the OS of that platform. Eg
  - On iOS, the plugin would be implemented in Switf using the native system services from iOS operating system
  - On Android, the plugin would be implemented in Java using the native system services from Android operating system

Application Packaging

- To develop a Cordova application, the application programmer writes the source code in HTML5, CSS and JavaScript.
- However the source code written by the application programmer alone is not sufficient.
- He also needs to copy the source code for each plugin he used in the application, eg, if the application made access to the camera, he would need to copy the source code for plugin `cordova-plugin-camera` in his application source code.
- This process is called application packaging.
- Packaging must be done for each platform you target your application to.

Building a Cordova Project

- Once the application has been packaged for the targeted platforms, you need to build the application for each targeted platform.
- Building an application for a platform means compiling the included plugins source code and linked the compiled code together to create an executable program.
- Note that the source code written by the application programmer needs no compilation as it consists of only HTML5, CSS and JavaScript, which can be directly executed by the web engine.
- The compilation will use the native development environment for each platform, eg,
  - iOS: Xcode
  - Android: Android Studio and SDK
  - Windows Phone: Microsoft Visual Studio for Windows Phone and SDK

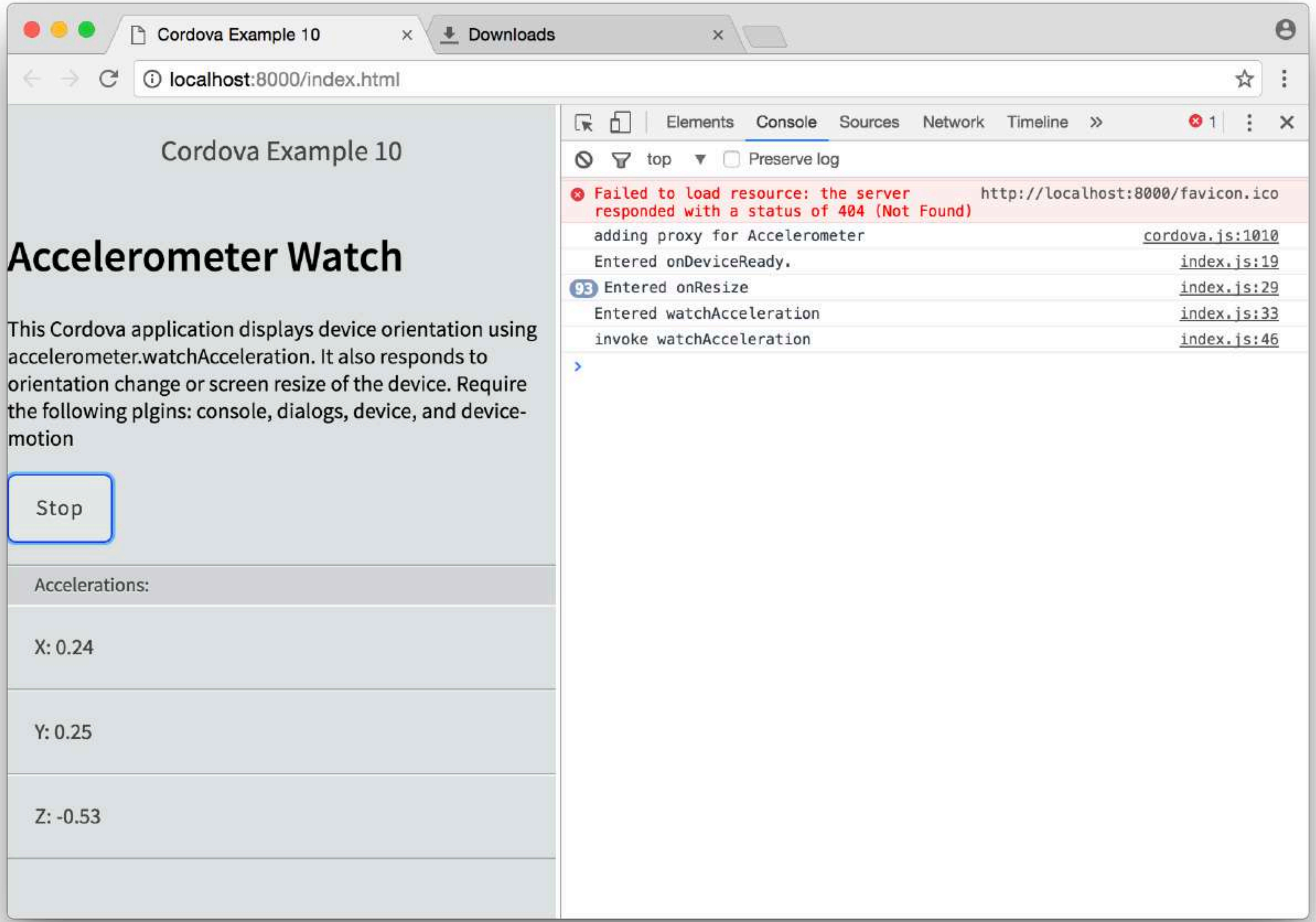
Testing an Application

- Once the application is built, you can install it and run it either in a device emulator or in a physical device.
- Xcode includes an emulator that could simulate various versions of iPhones and iPads.
- Android Studio provides an emulator that can simulate several different types of Android devices at different API levels. The latest API level is 29.
- You can also install and run an application directly on a physical device.
- If you use Windows to develop the application, you may need to install the USB driver from the device manufacturer before you can run the application on the device, and you must also set the device to USB debugging mode.

Testing an Application on Browser Platform

- As a Cordova application is written in HTML5, CSS and JavaScript, it can be tested on a web browser (although some hardware features may not be available on a web browser).
- In many cases it is much easier to test the application in a web browser first, as browsers such as Google Chrome provide very good debugging tools for web applications (View | Developer | JavaScript Console).
- It is preferably that you test your application with a browser first to fix many problems related HTML, CSS and JavaScript, before testing it on an emulator or physical device.

Testing an Application on Browser Platform



Cordova Example 10

Accelerometer Watch

This Cordova application displays device orientation using accelerometer.watchAcceleration. It also responds to orientation change or screen resize of the device. Require the following plgins: console, dialogs, device, and device-motion

Stop

Accelerations:

X: 0.24

Y: 0.25

Z: -0.53

```
Failed to load resource: the server      http://localhost:8000/favicon.ico
responded with a status of 404 (Not Found)
adding proxy for Accelerometer                          cordova.js:1010
Entered onDeviceReady.                                    index.js:19
93 Entered onResize                                       index.js:29
Entered watchAcceleration                                 index.js:33
invoke watchAcceleration                                  index.js:46
```

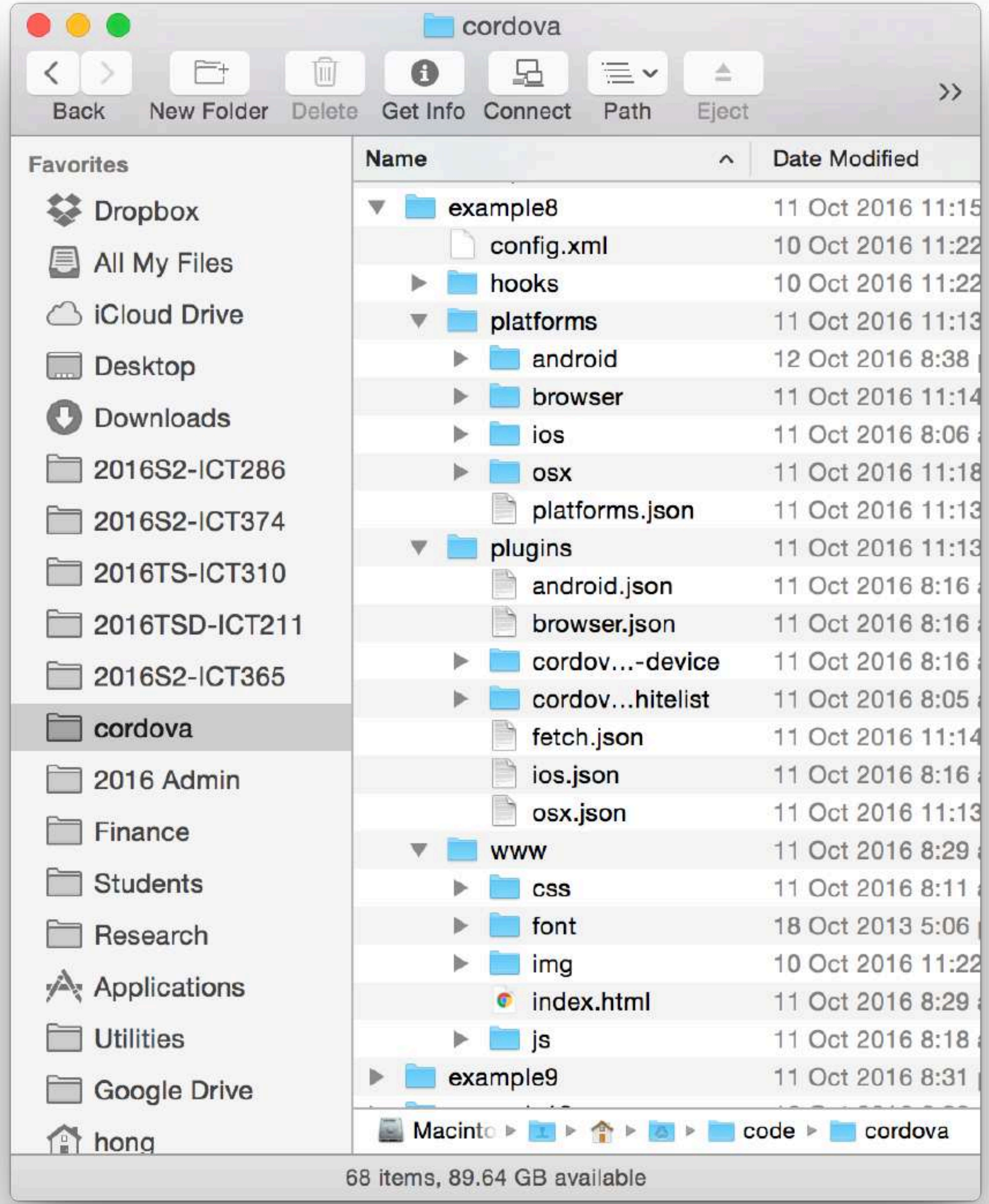
Cordova Command-Line Interface (CLI)

- Apart from the library that provides a consistent JavaScript API for accessing hardware and system features, Cordova also provides a Command-Line Interface, or CLI.
- CLI allows us to develop a Cordova application (eg, packaging, building, testing, and running an application) in a consistent way whether we use Microsoft Windows, Mac, or Linux.
- Some of the CLI commands:
  - cordova create hello com.ict286.hello HelloWord
  - cordova platform add ios android browser
  - cordova requirements android
  - cordova plugin add cordova-plugin-device
  - cordova build
  - cordova run browser
  - cordova run android --target=Nexus_5_23
  - cordova run ios --target=iPhone-6

Cordova Project Folder

- A Cordova project directory consists of a number of sub-directories:
  - www: this directory contains the HTML5, CSS, and JavaScript code written by the application programmer.
  - platforms: contains the packaged source code for the application and built executable for each targeted platform
  - plugins: contains the source code for each plugin added to the application.
- Application programmer should not manually modify anything under platforms and plugins sub-directories. The code in these two sub-directories are created by Cordova commands.
- Application programmer usually places his HTML code in file `www/index.html`, CSS code in file `www/css/index.css`, and JavaScript code in file `www/js/index.js`.

Cordova Project Folder



Name	Date Modified
▼ example8	11 Oct 2016 11:15
config.xml	10 Oct 2016 11:22
▶ hooks	10 Oct 2016 11:22
▼ platforms	11 Oct 2016 11:13
▶ android	12 Oct 2016 8:38
▶ browser	11 Oct 2016 11:14
▶ ios	11 Oct 2016 8:06
▶ osx	11 Oct 2016 11:18
platforms.json	11 Oct 2016 11:13
▼ plugins	11 Oct 2016 11:13
android.json	11 Oct 2016 8:16
browser.json	11 Oct 2016 8:16
▶ cordov…-device	11 Oct 2016 8:16
▶ cordov…hitelist	11 Oct 2016 8:05
fetch.json	11 Oct 2016 11:14
ios.json	11 Oct 2016 8:16
osx.json	11 Oct 2016 11:13
▼ www	11 Oct 2016 8:29
▶ css	11 Oct 2016 8:11
▶ font	18 Oct 2013 5:06
▶ img	10 Oct 2016 11:22
index.html	11 Oct 2016 8:29
▶ js	11 Oct 2016 8:18
▶ example9	11 Oct 2016 8:31

68 items, 89.64 GB available

So What is Apache Cordova?

- Apache Cordova is a set of libraries and tools that allow us to create cross-platform mobile applications using HTML5, CSS and JavaScript from Windows, Mac and Linux that targets all major mobile operating systems.
  - Cordova provides the same set of JavaScript API to access the underlying hardware and system features on the supported mobile operating systems;
  - Cordova provides the same software development environment (eg, CLI) to develop mobile apps on Windows, Mac and Linux;
  - Cordova applications are native applications – the access to the underlying hardware and system features is done by the code native to the underlying operating system;
  - Cordova applications are written in the common web languages: HTML5, CSS and JavaScript.

Cordova vs Native Development Environments

- Android Studio and SDK:
    - application code is written in Java
    - screen drawing is done using UI related Java libraries
    - access to the operating system service is done in Java
    - the application can only run on Android devices
- Xcode:
    - application code is written in Swift
    - screen rendering is done using UI related libraries
    - access to the operating system service is done in Swift
    - the application can only run on iOS devices, ie, iPhones and iPads
- Apache Cordova:
    - application code is written in HTML5, CSS, and JavaScript
    - page rendering is done by the same web engine used by the device's default web browser
    - access to the operating system service is done in JavaScript
    - the application, once properly packaged and built, runs on all supported mobile operating systems

Pros and Cons

- Universality and Consistency
  - Using native development environments, you need to develop one application for each mobile operating system
    - Not universal (not cross-platform)
    - May not behave consistently on different platforms as they are different applications
  - With Cordova, one application runs on (nearly) all operating systems
    - Universal (ie, cross-platform)
    - Behave consistently on all platforms
- Development cost
  - Native environments: need to learn multiple environments, multiple OS and multiple languages
    - High cost
  - Cordova: one application for all platforms, and HTML5, CSS and JavaScript are common web technologies
    - Low cost

Pros and Cons

- Efficiency
  - With a native development environment, the application could be as efficient as the operating system allows
  - With Cordova, it could be argued that as JavaScript is an interpreted language it could be less efficient than a compiled language native to the operating system (eg, Swift)
    - However one must remember that many web engines use Google's JavaScript engine: V8 engine, which is compiled and is very efficient
    - Many games are written with Cordova – this shows that a well-written Cordova application could be as efficient as the application written in a native language.
- Access to operating system services
  - Native environments: an application can access any service available from the underlying operating system
  - Cordova: some available operating system services may not yet be available as Cordova plugins, hence not available to the application programmer directly
    - However, there are established procedures and protocols for anyone with enough expertise to write a new Cordova plugin to incorporate the service, hence this is not a fundamental obstacle.

Pros and Cons

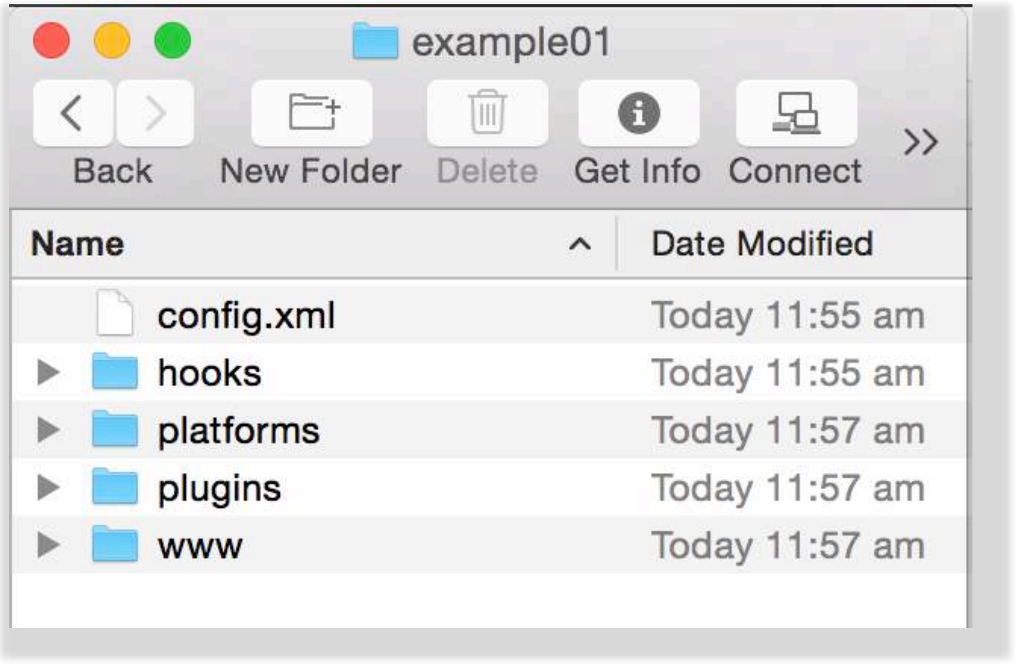
- Easiness in creating graphical user interface
  - Most native development environments, such as Xcode, Visual Studio, and to some extent, Android Studio, provides excellent tools for you to design graphical user interface of your application using “drag and draw”. the source code for the GUI will be generated automatically by the tool
  - Cordova does not provide such a “drag and draw” tool for you to design your graphical user interface.
    - However, as the languages used to write the graphical user interface are the common web languages (HTML5, CSS and JavaScript), there are plenty of web page authoring tools such as Adobe DreamWeaver, Microsoft Expression Web Designer for this purpose.
    - there are a number of JavaScript frameworks for developing mobile apps that greatly simplify the UI design, such as jQuery Mobile, Ionic and Boostrap.

First Cordova Application

- Open up a terminal (eg, Command Prompt on Windows, terminal on Mac and Linux)
- Create a new cordova project:

```
cordova create example01 com.ict286.example1 Example1
```

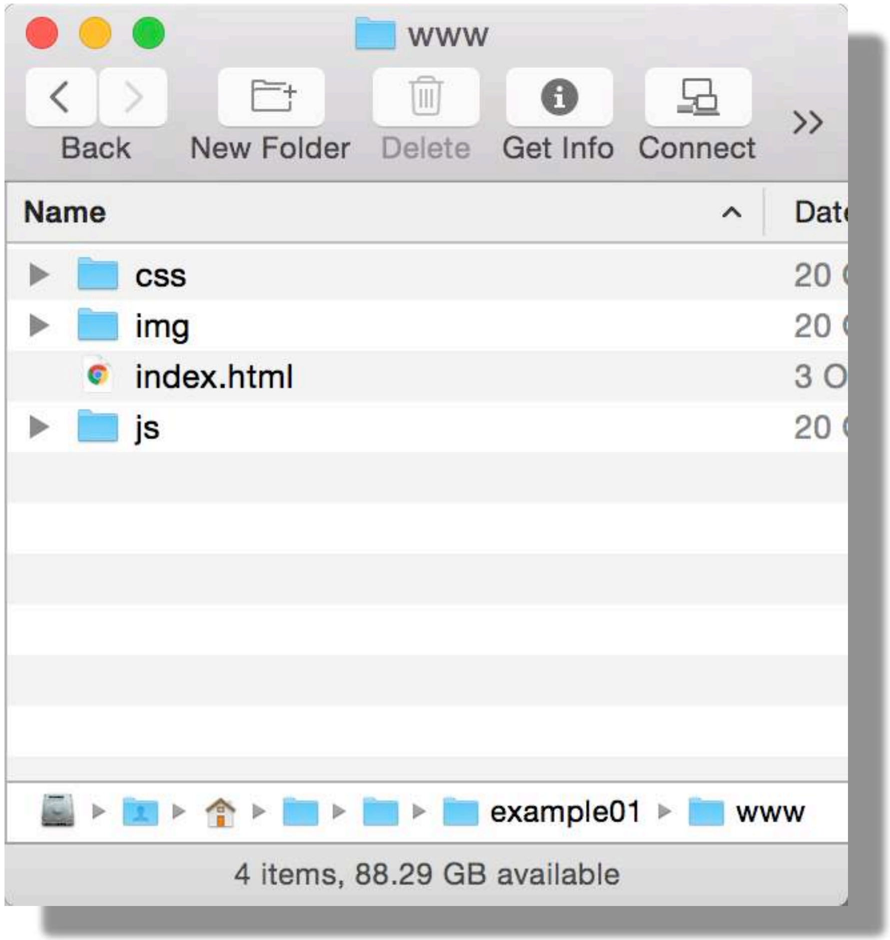
You will create a new project folder named “example01” in your current directory.



Name	Date Modified
config.xml	Today 11:55 am
hooks	Today 11:55 am
platforms	Today 11:57 am
plugins	Today 11:57 am
www	Today 11:57 am

The `www` Folder

- You do not need to manually modify any files or directories under project folder except the files in www directory. The default structure of the www directory looks like below:



css 20
img 20
index.html 3 O
js 20

example01 ▸ www

4 items, 88.29 GB available

Files under `www` Folder

- The most important file in the WWW directory is the file `index.html`. You need to place your application's HTML code there.
- By convention, you should place your CSS style sheets under `www/css` directory. The default file name of your style sheet is `index.css`.
- By convention, you should place your JavaScript code under `www/js`. The default file name for the JavaScript code is `index.js`.
- Similarly, place images under `www/img`.
- Our first application doesn't use any style sheet or JavaScript code. Hence you may remove those files or leave them there untouched. Our HTML code would not make any reference to them anyway!

HTML Code

- Use a text editor to change the content of the `index.html` file to:

```
<!DOCTYPE html>
<html>
<head>
   <title> Cordova Example 1 </title>
</head>
<body>
   <h1> Cordova Example 1 </h1>

   <p>This is our first Cordova example. This is just a
web application. </p>

</body>
</html>
```

Package and Build

- As you can see our first Cordova application is just a web application, or more precisely, just a simple HTML page!
- It is true that you can turn any web page, or set of web pages into a Cordova application
  - You don’t have to use JavaScript
  - You don’t have to use any Cordova service
  - But you do need to package it for the target platforms and you do need to build the application!
- Packaging and building:

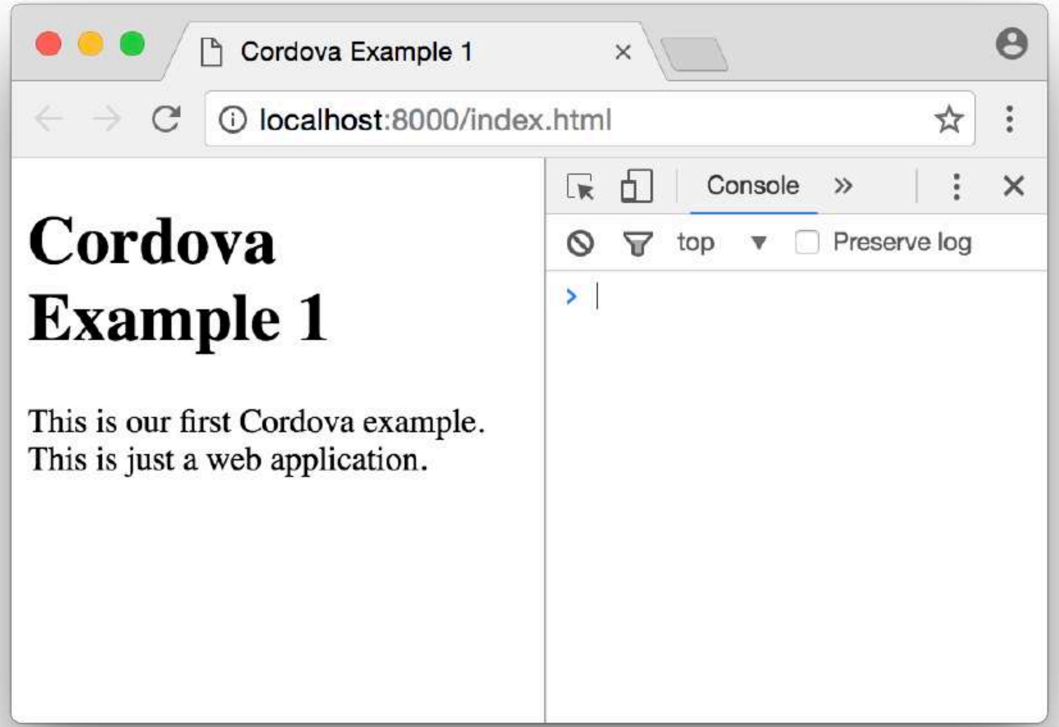
```
cordova platform add android ios browser
cordova build
```

  - The first command adds the targeted platforms to the project – in this case, Android, iOS, and browser (so that you can also test it in a web browser).
  - The second command packages and builds the application for all three platforms.
  - Note: you must make sure that you are in the project folder before typing any cordova command for that project! Use cd command to enter a project folder.

Test the App in a Browser

- Once the application is successfully built, you can run it in any one of the targeted platforms.
- Test it in a web browser (the right pane is the JavaScript console, which can be turned on or off in Google Chrome):

```
cordova run browser
```



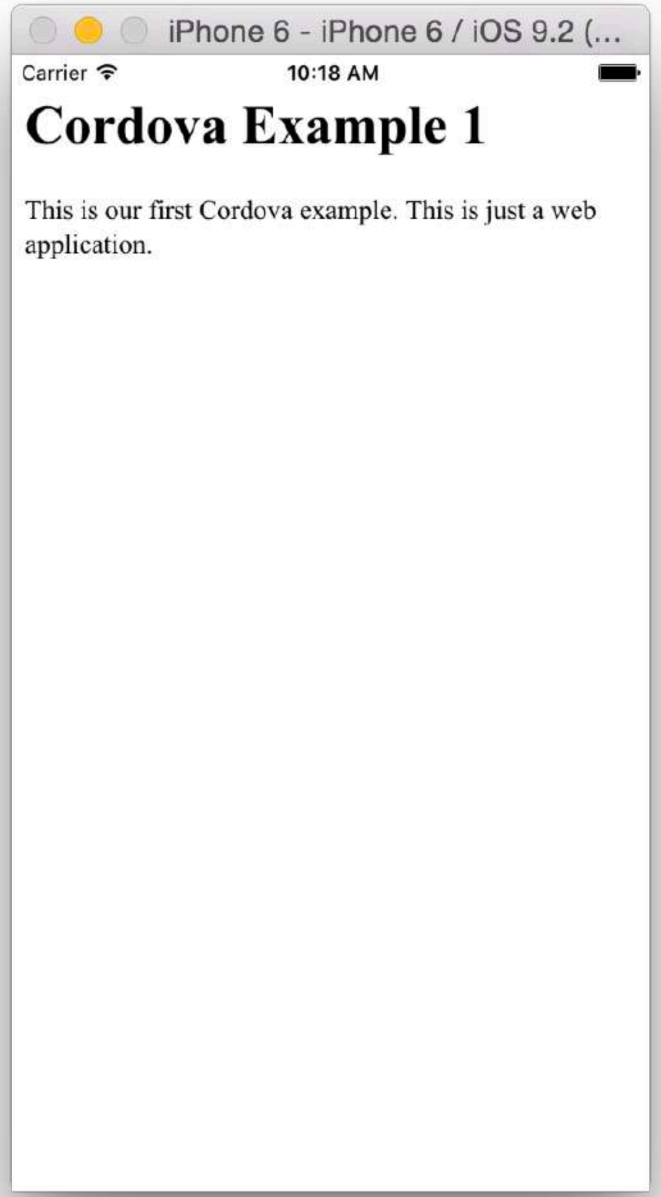
Cordova Example 1

This is our first Cordova example. This is just a web application.

Test the App on iOS

- Of course you can test it in an emulator or even on a physical device.
- Test it on an iPhone emulator

```
cordova run ios
    --target=iPhone-6
```



Cordova Example 1

This is our first Cordova example. This is just a web application.

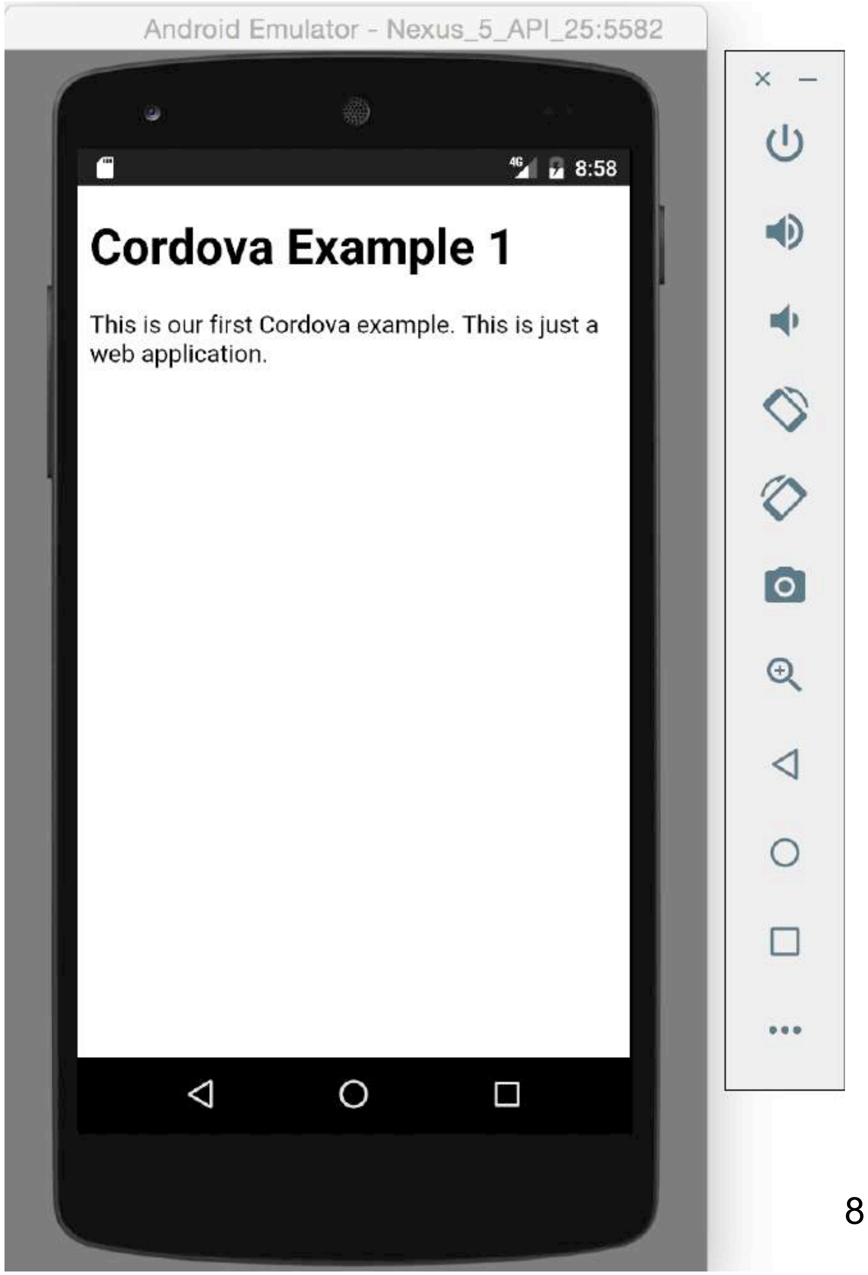
Test the App on Android

- Or on an Android emulator

```
cordova run android
    --target=Nexus_5_API_25
```

- You can find out what emulators or devices are available on your machine and their names using the following command

```
cordova run --list
```



Cordova Example 1

This is our first Cordova example. This is just a web application.

Example 2

- Being a Cordova application, naturally most of times it would need to use Cordova services (plugins).
- However it cannot use any Cordova service before Cordova is initialised. Once Cordova finishes its initialisation, it fires off a “deviceready” event.
- Our second example illustrates how to capture this event and then use one of the Cordova plugins to display a message to inform the user that Cordova is ready.
- This application needs to firstly register the event handler for “deviceready” event.
- Once the event is fired, the event handler is called which will call `navigator.notification.alert` method to display a message. This `notification` object comes from `cordova-plugin-dialogs`.

The Source Code

- `www/index.html`

```
<!DOCTYPE html>
<html>
<head>
    <title>Cordova Example 2</title>
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width" />
     <script src="cordova.js"></script>
     <script>
        function onBodyLoad() {
           console.log("Entered onBodyLoad");
           document.addEventListener("deviceready", onDeviceReady, false);
        }
        function onDeviceReady() {
           // Cordova finishes initialisation
           console.log("Cordova is ready");
           // need to add dialogs plugin:
           // cordova plugin add cordova-plugin-dialogs
           navigator.notification.alert("Cordova is ready!");
        }
     </script>
</head>
```

The Source Code

- `www/index.html` (continue)

```
<body onload="onBodyLoad()">

    <h1>Cordova Example 2</h1>
    <p>This is the second Cordova example. It displays a dialog when Cordova
is ready.
    Require plugin dialogs.</p>

</body>
</html>
```

The Viewport Meta Tag

- The viewport is the user’s visible area of a web page. It may be or may be not equal to the screen area.
- If the view is larger than the screen, the user may need to move the page left and right, up and down, in order to see the whole page.
- Or the user may need to scaled it down to see the whole page (but everything would be smaller), or to scaled it up to see small prints.
- Please note that the handling of viewport meta is inconsistent among different devices. For most mobile apps, use the following attributes:

-
  ```
  <meta name="viewport" content="user-scalable=no,
  initial-scale=1, maximum-scale=1, minimum-
  scale=1, width=device-width" />
  ```

Import Cordova Library

- To use Cordova services the application must use Cordova library by providing the following reference

```
<script src="cordova.js"></script>
```

- This script element could be placed inside head element or at the end of the body element.
- Although we have placed the two JavaScript elements inside the head element in this application, this is not necessary.
- It is actually preferrable to place the JavaScript code at the end of the body element so that loading and rendering of HTML code is not delayed by loading large JavaScript code – better user experience.

Register an Event Handler

- In this application there are two JavaScript functions in addition to the Cordova library.
- The first function, `onBodyLoad`, defines an event handler (or callback) for *`onload`* event.

```
function onBodyLoad() {
    console.log("Entered onBodyLoad");
    document.addEventListener("deviceready",
        onDeviceReady, false);
}
```

  – Note, `addEventListener` was introduced in DOM Level 3 and it allows us to define whether the event handling would be "bubbling up" (3rd argument `false`) or "capturing" (3rd argument `true`).

- This callback sends a message to the console (which can be seen from JavaScript console if you test it in a browser – good for debugging JavaScript code) immediately after the HTML page is loaded.
- It then registers an event handler (a function named `onDeviceReady`) for another event – *`deviceready`* event, using `addEventListener` method from `Document` object.

Get Cordova Services

- The second function, `onDeviceReady`, defines an event handler for “deviceready” event.

```
function onDeviceReady() {
    console.log("Cordova is ready");
    // need to add dialogs plugin:
    // cordova plugin add cordova-plugin-dialogs
    navigator.notification.alert("Cordova is ready!");
}
```

- Once Cordova finishes its initialisation, the `deviceready` event would be fired, and the above event handler would be called automatically.
- You can then start to get services from Cordova plugins. In this case we just display a message to inform the user that Cordova is ready for action.
  - Although the same thing could be achieved with the `alert` method from `Window` object, we deliberately choose to use the `alert` method from Cordova to make a point here.

Package, Build and Run the Application

- The project Example2 can be packaged and built with the following commands:
  - add the targeted platforms to the project. In this case we target the app to iOS, Android and browser:

    ```
    cordova platform add ios android browser
    ```
  - add the required plugins into the project – this app has only used `navigator.notification` object, which is provided by the dialogs plugin, hence

    ```
    cordova plugin add cordova-plugin-dialogs
    ```
  - then build the project for all targeted platforms:

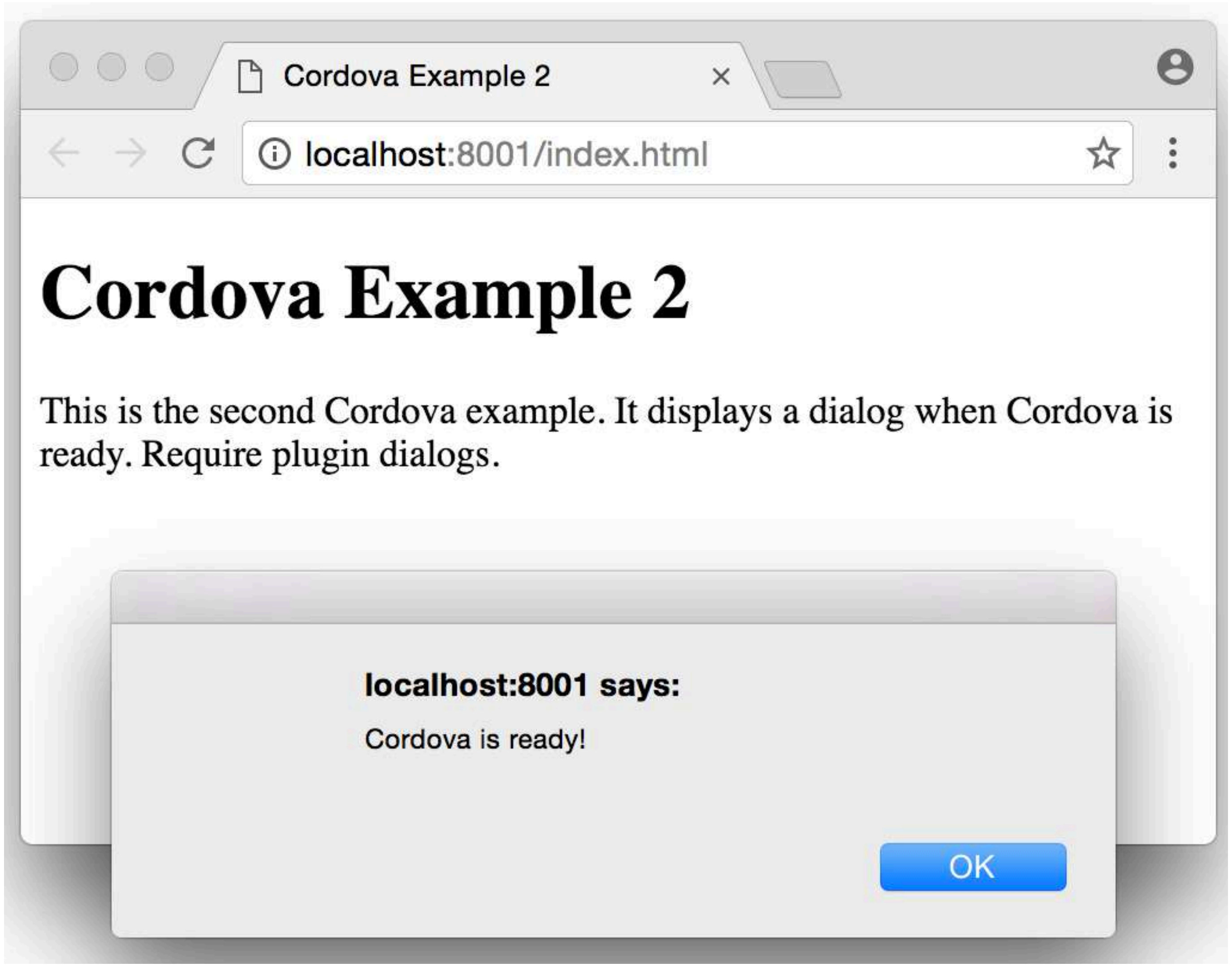
    ```
    cordova build
    ```
  - to test the app in a browser, try

    ```
    cordova run browser
    ```
  - to run it in an iOS device named iPhone-6 (actually an emulator), try

    ```
    cordova run ios --target=iPhone-6
    ```
  - to run it in an Android device named Nexus_5_API_24 (an emulator): try

    ```
    cordova run android --target=Nexus_5_API_24
    ```

Test Example 2

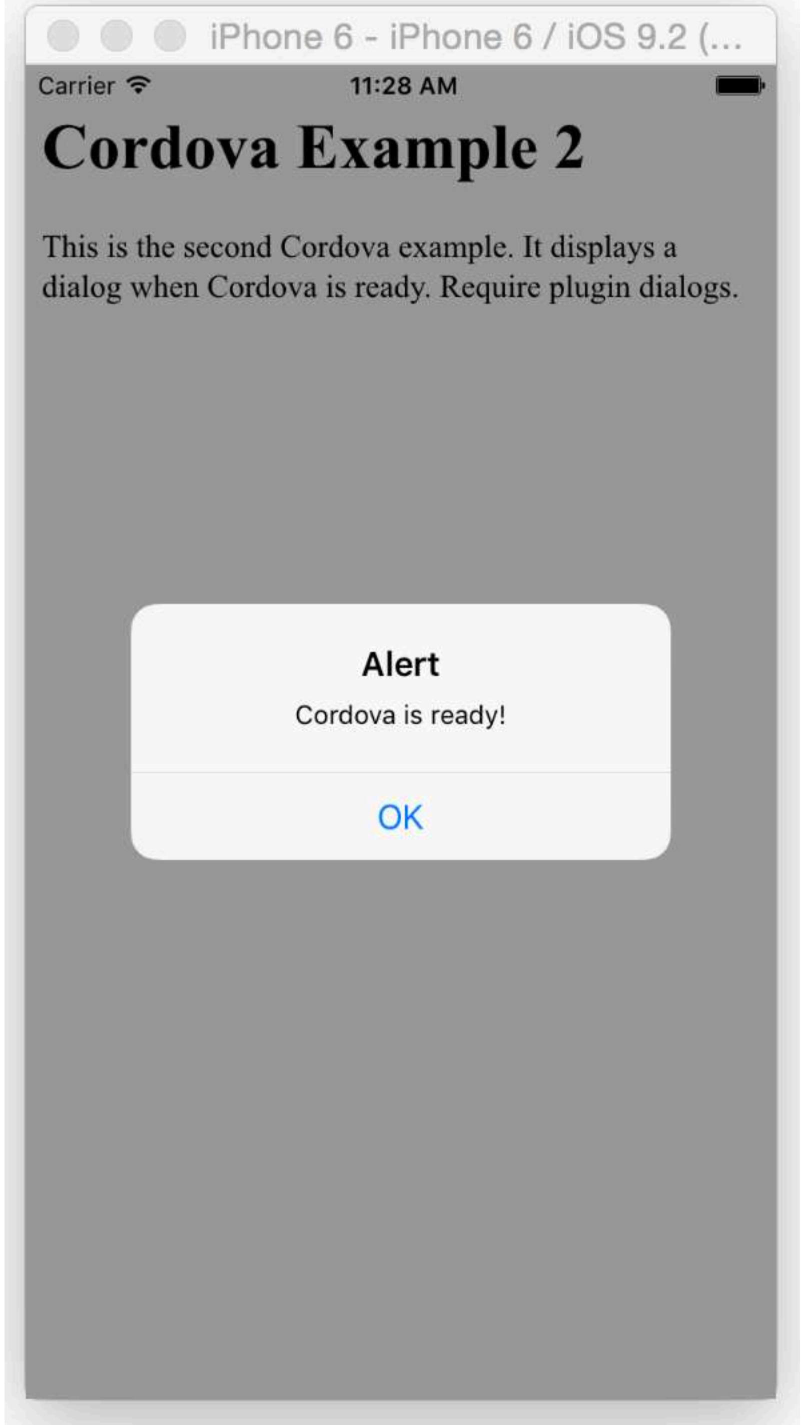


Cordova Example 2

This is the second Cordova example. It displays a dialog when Cordova is ready. Require plugin dialogs.

localhost:8001 says:

Cordova is ready!

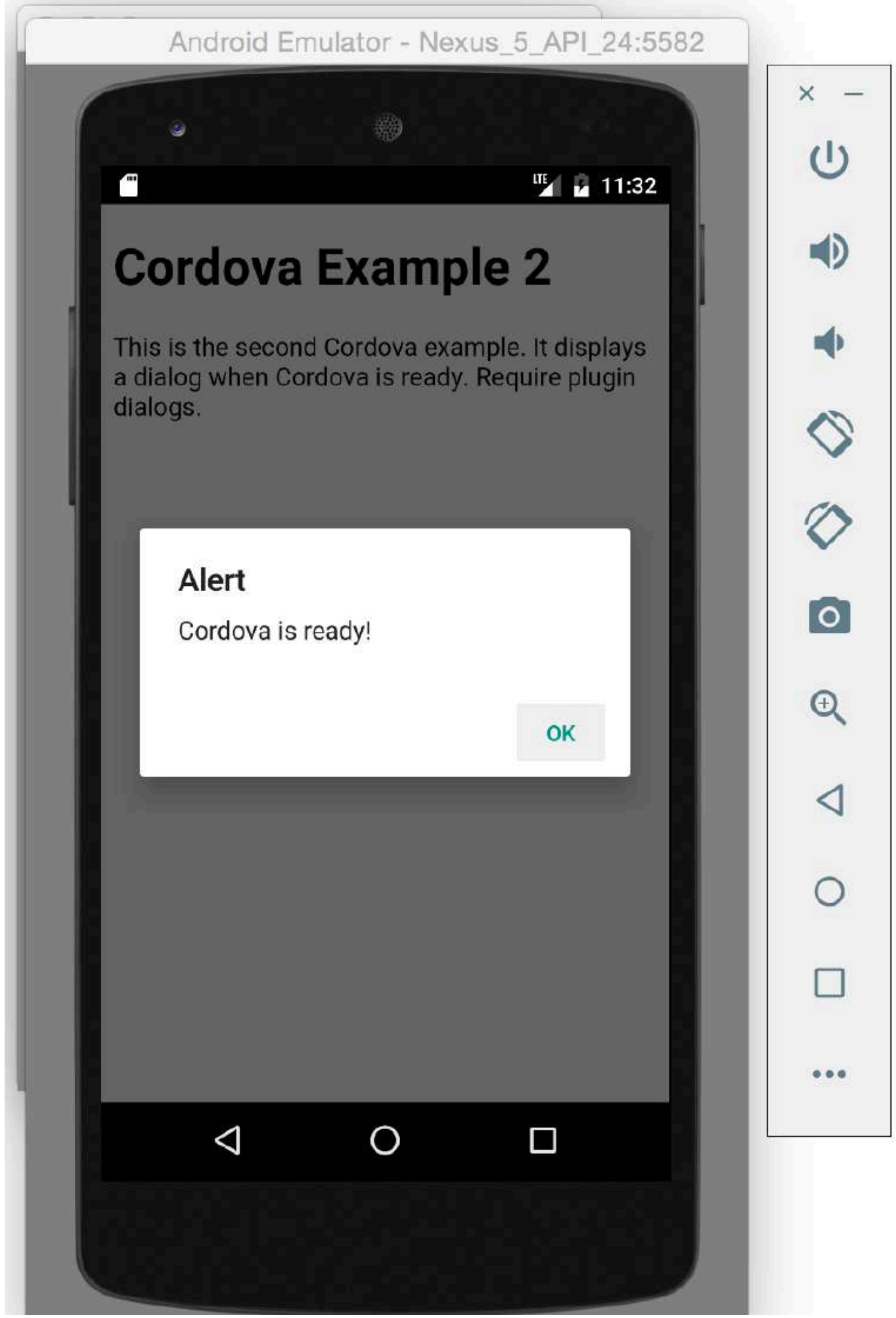


Cordova Example 2

This is the second Cordova example. It displays a dialog when Cordova is ready. Require plugin dialogs.

Alert

Cordova is ready!



Cordova Example 2

This is the second Cordova example. It displays a dialog when Cordova is ready. Require plugin dialogs.

Alert

Cordova is ready!

Example 3

- In the third example, we show how files are named, and where the JavaScript code is placed in a typical Cordova app. Firstly let’s look at the head of the `index.html` like:

```
<!DOCTYPE html>
<html>
<head>
    <title>Cordova Example 3</title>
    <meta name="viewport" content="user-scalable=no, initial-
scale=1, maximum-scale=1, minimum-scale=1, width=device-width" />
    <link rel="stylesheet" type="text/css" href="css/index.css" />
</head>
```

- Here we refer to a stylesheet named `index.css` which is placed under `css` folder.

Style Sheet

- The CSS file `index.css` looks like this:

```
img {
    position: absolute;
    left: 0px;
    top: 0px;
    z-index: -1;
}
h1 {
    position: relative;
    left: 55px;
    top: -10px;
}
```

- Which places the image at absolute position at the top left corner of the viewport, and places the header h1 at relative position, so that the Codova logo and the header are vertically aligned.

Placement of JavaScript Code

- Next we take a look at the body of the `index.html` file:

```
<body onload="onBodyLoad()">
    <h1> Example 3 </h1>
    <img src="img/logo.png" alt="Cordova Logo" height="60" />

    <p>This Cordova example displays the device information.
       Require cordova-plugin-device. </p>

    <p id="appInfo"> </p>

     <script src="cordova.js"></script>
     <script src="js/index.js"></script>
</body>
</html>
```

- As you can see, we will load JavaScript code at the end. The user defined JavaScript code is placed in the file `js/index.js`.

JavaScript Code

- Next we have a look at the JavaScript code from file js/index.js:

```
function onBodyLoad() {
    console.log("Entered onBodyLoad");
    document.addEventListener("deviceready", onDeviceReady,
false);
}

function onDeviceReady() {
    console.log("Entered onDeviceReady");
    // report App info
    var info = document.getElementById("appInfo");
    info.innerHTML =
            "Cordova version: " + device.cordova + "<br/>"
          + "Platform: "        + device.platform + "<br/>"
          + "Model: "           + device.model + "<br/>"
          + "OS version: "      + device.version;
}
```

Get Device Information

- The callback function `onBodyLoad` would register the event handler for *`deviceready`* event once the HTML code is loaded.
- Once Cordova finishes its initialisation, it would fire off the event *`deviceready`*, resulting in calling `onDeviceReady` callback.
- In the callback, we can start using Cordova services.
- In this app, we simply use `cordova.plugin.device` to obtain some information about Cordova and the device.
- We then construct an HTML fragment on the fly and place it under the paragraph with id `appInfo`, using `innerHTML` property.

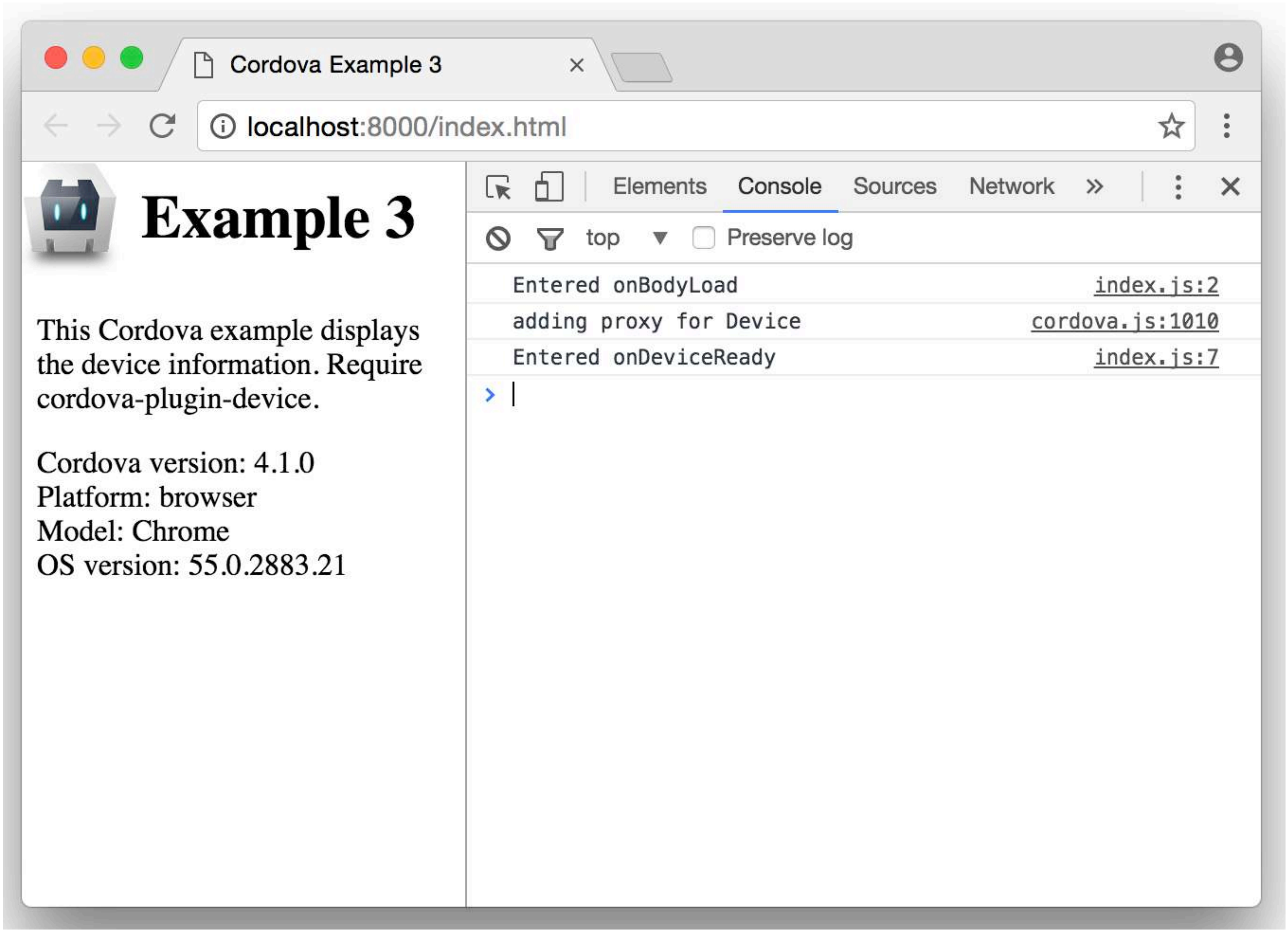
Build and Run

- Example 3 can be created, packaged, built and tested using the following CLI commands.

```
cordova create example03 com.ict286.example3 Example3
cd example03
```

Create the HTML code, CSS code and JS code for the application

```
cordova platform add browser ios android
cordova requirements
cordova plugin add cordova-plugin-device
cordova build
cordova run --list
cordova run browser
cordova run ios --target=iPhone-6
cordova run android --target=Nexus_5_API_25
```

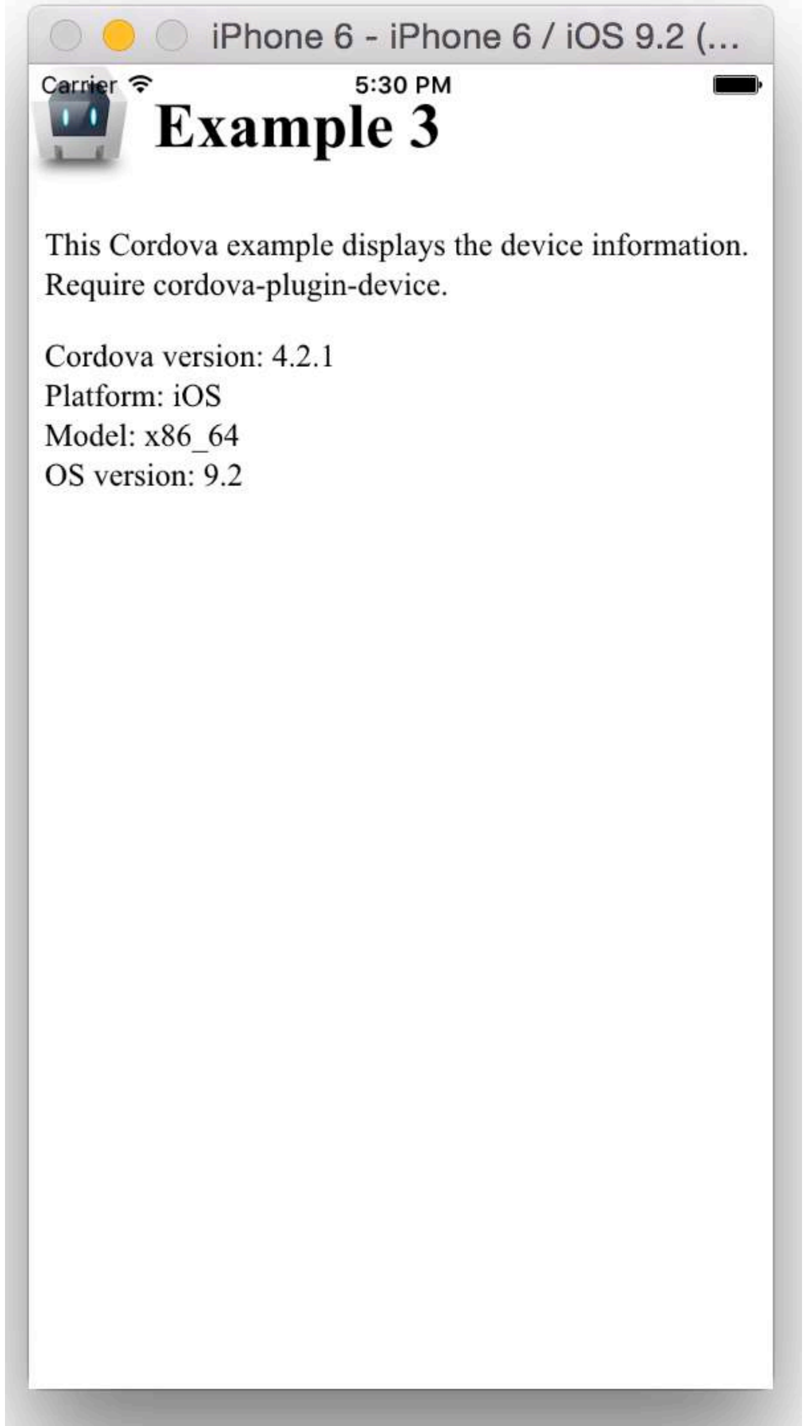


Example 3

This Cordova example displays the device information. Require cordova-plugin-device.

Cordova version: 4.1.0
Platform: browser
Model: Chrome
OS version: 55.0.2883.21

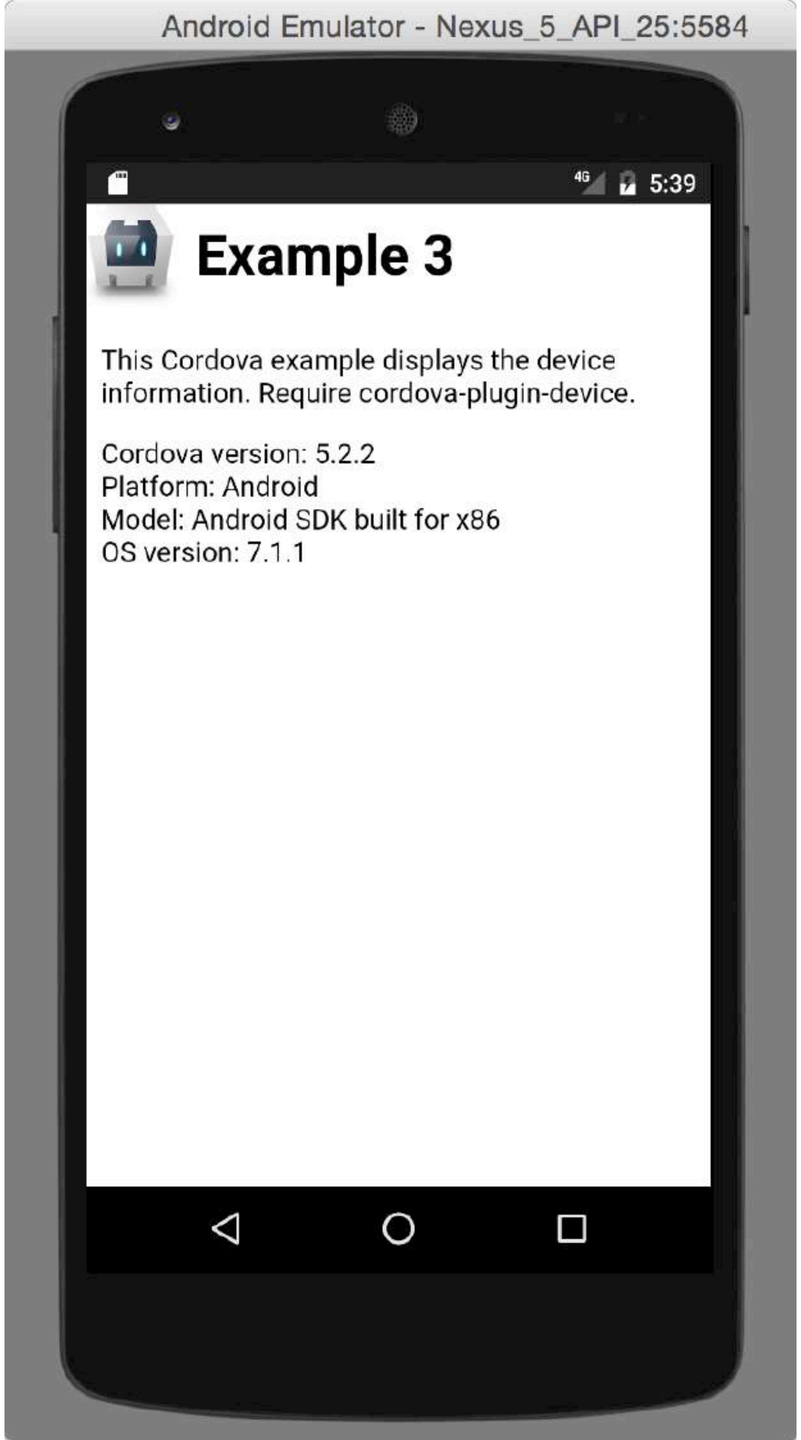
```
Entered onBodyLoad                          index.js:2
adding proxy for Device                  cordova.js:1010
Entered onDeviceReady                       index.js:7
```



Example 3

This Cordova example displays the device information. Require cordova-plugin-device.

Cordova version: 4.2.1
Platform: iOS
Model: x86_64
OS version: 9.2



Example 3

This Cordova example displays the device information. Require cordova-plugin-device.

Cordova version: 5.2.2
Platform: Android
Model: Android SDK built for x86
OS version: 7.1.1

Example 4

- Our fourth example is identical to Example 3 in functionality, however the code is structured in a typical Cordova way!
- It is highly recommended that you follow this structure when you design your Cordova apps.
- The code in `www/index.html`, `www/css/index.css` is identical to those in the same files in Example 3.
- The code in `www/js/index.js`, although perform the same task, looks very different from the JavaScript code in Example 3!
- In this code, we define a JavaScript object that consists of a set of *`variable`*`:`*`value`* mapping, which is a JavaScript object literal.

JavaScript Code in `js/index.js`

```
var app = {
    initialize: function() {
        this.bindEvents();
    },

    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },

    onDeviceReady: function() {
        app.updatePage();  // would this.updatePage() work?
    },

    updatePage: function() {
        var info = document.getElementById("appInfo");
        info.innerHTML = "Cordova version: " + device.cordova + "<br/>"
                       + "Platform: "        + device.platform + "<br/>"
                       + "Model: "           + device.model + "<br/>"
                       + "OS version: "      + device.version;
    },
};

app.initialize();
```

JavaScript Object Literal

- Before we explain the JavaScript program in Example 4, we need to have another look at JavaScript objects.
- In JavaScript, pretty much everything is an object.
- There are many ways we can define an object. We have learnt a few ways in Topic 3.
- One of the ways to define a JavaScript object is use object literal:

```
var obj = {
     name1: value1,
     name2: value2,
     . . . . . .
     namen: valuen
}
```

- In this way, we define and create a single object (an object literal), named obj.

JavaScript Object Literal

- Each `name:value` pair defines a property of the object. You can access the property of an object by its name: `obj.name`. Eg,

```
var unitName = {
   ict286: "Web Computing",
   ict365: "Software Development Frameworks",
   ict375: "Advanced Web Programming"
};
console.log("old name: " + unitName.ict286);
unitName.ict286 = "Web and Mobile Computing";
console.log("new name: " + unitName.ict286);
```

- You can test above code using `node` from a terminal, as in Topic 3. Place the above code in a file such as `test.js`, and type command

```
node test.js
```

(don’t be lazy – type the code in, do not copy-and-paste ☺)

The Method of an Object

- However in a `name:value` pair, the value doesn't have to be a string or a number. It can also be a function (or even another object)! In this case, the function is also known as a method of the object. See the following example:

```
var circle = {
     radius: 1,
     setRadius: function(radius) {
          this.radius = radius;
     },
     area: function() {
          return Math.PI * this.radius * this.radius;
     },
     circumference: function() {
          return Math.PI * this.radius * 2;
     }
}
```

What is `this`?

- Note that in the method `setRadius`, `area`, and `circumference`, the use of `this` to access a property in the same object. Eg,

```
area: function() {
    return Math.PI * this.radius * this.radius;
},
```

- As property `radius` belongs to an object, you have to use *`object.name`* to access the property, unless the property is defined in the global scope.
- The reserved word `this` represents the object the function is *currently* in.
- In the above example, function `area` is currently in object `circle`, hence `this.radius` is equivalent to `circle.radius`. The above return statement could be written as

```
return Math.PI * circle.radius * this.radius;
```

What is `this` Again

- Now back to object `app` defined in file `js/index.js` earlier. In the callback `onDeviceReady`, we invoke `updatePage` method with

  ```
  app.updatePage();
  ```

  Would the callback work if we invoke `updatePage` with either

  ```
  updatePage();
  ```

  or

  ```
  this.updatePage();
  ```

- The answer is No!

What is `this` Again

- The first statement, `updatePage();`, is incorrect. As it refers to a function defined in the global scope, where no such function is defined.
- The second statement, `this.updatePage();`, could be right *if* `this` refers to object `app` at the time the function is called.
- Unfortunately this is not the case!
- Remember the callback is not invoked inside the object `app`. Instead it is invoked in the event loop - a completely different scope from object `app` - when the relevant event is triggered at a future point of time.
- Hence, the reserved word `this` represents the object in which the callback is invoked, not object `app`!
- Therefore the second statement would not work!

Example 5

- Our fifth example shows how to detect changes in device orientation and size. When the change is detected, the new screen size and browser’s inner window size are displayed (see `example05/www/js/index.js` for complete code).

```
updatePage: function() {
    var orient = "<strong>Orientation: </strong>"
           + window.orientation + "degree <br/>";
    var swidth = "<strong>Screen width: </strong>"
           + screen.width + "<br/>";
    var sheight = "<strong>Screen height: </strong>"
           + screen.height + "<br/>";
    var wwidth = "<strong>Window inner width: </strong>"
           + window.innerWidth + "<br/>";
    var wheight = "<strong>Window inner height: </strong>"
           + window.innerHeight + "<br/>";
    document.getElementById("appInfo").innerHTML
           = orient+swidth+sheight+wwidth+wheight;
},
```

Events: orientationchange and resize

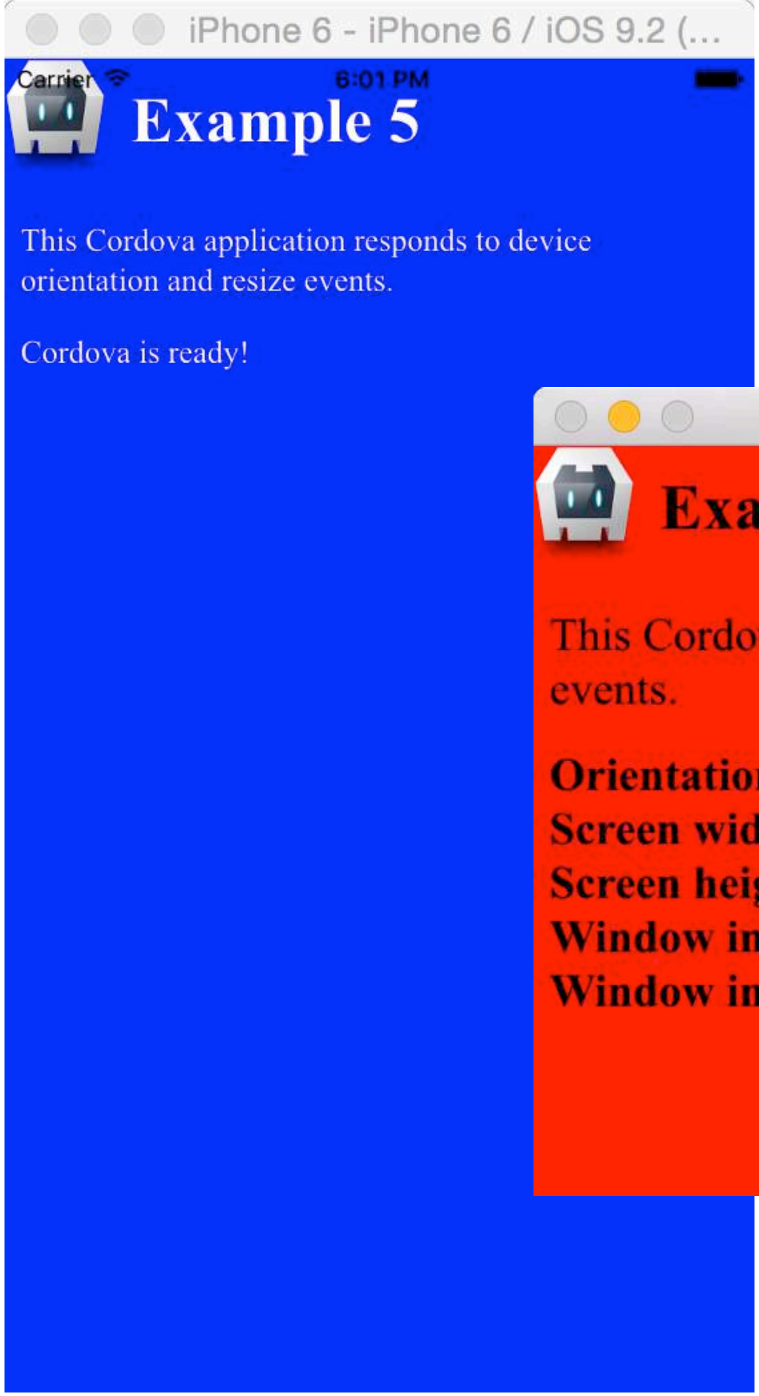
- The program registers two window events: `orientationchange` and `resize`.

```
bindEvents: function() {
    document.addEventListener('deviceready',
              this.onDeviceReady, false);
    window.addEventListener('orientationchange',
              this.onOrientationChange);
    window.addEventListener('resize', this.onResize);
},

onOrientationChange: function(id) {
    app.updatePage();
},

onResize: function(id) {
    app.updatePage();
}
```

Media Query

- The program also uses media query introduced in CSS3 to apply different styles in portrait and landscape (see `example05/www/css/index.css`).

```
@media screen and (orientation: portrait) {
   body {
        background-color: blue;
        color: white;
   }
}

@media screen and (orientation: landscape) {
   body {
        background-color: red;
        color: black;

   }
}
```

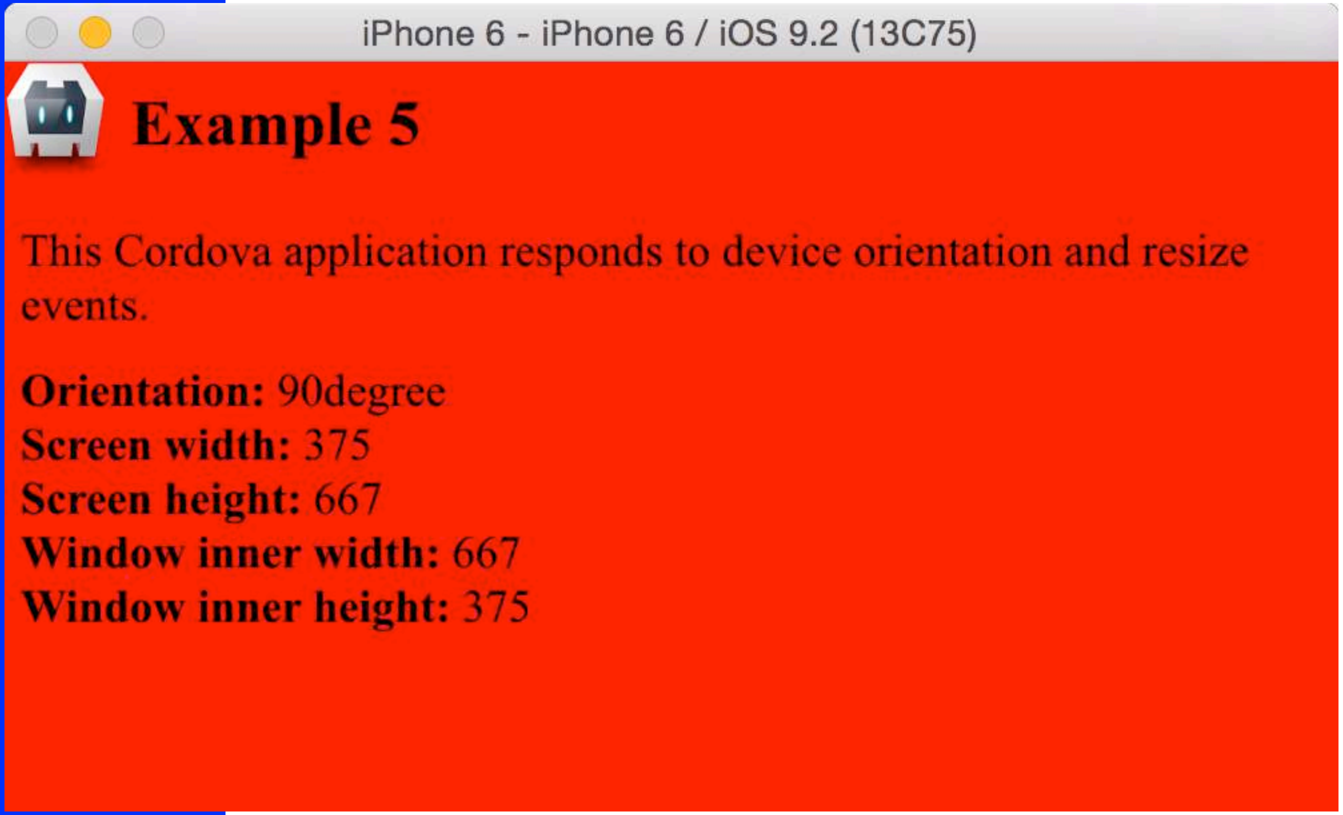

iPhone 6 - iPhone 6 / iOS 9.2 (...

Carrier 6:01 PM

Example 5

This Cordova application responds to device orientation and resize events.

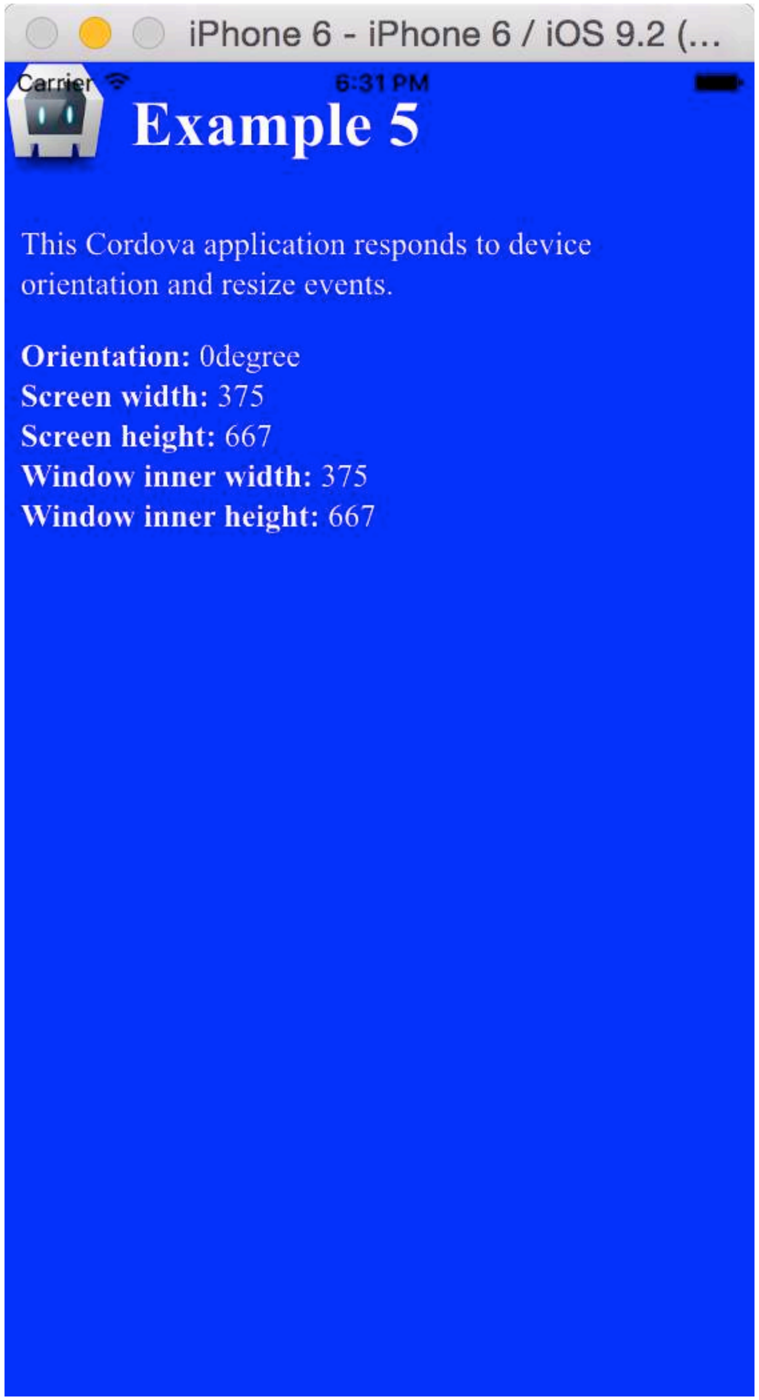
Cordova is ready!


iPhone 6 - iPhone 6 / iOS 9.2 (13C75)

Example 5

This Cordova application responds to device orientation and resize events.

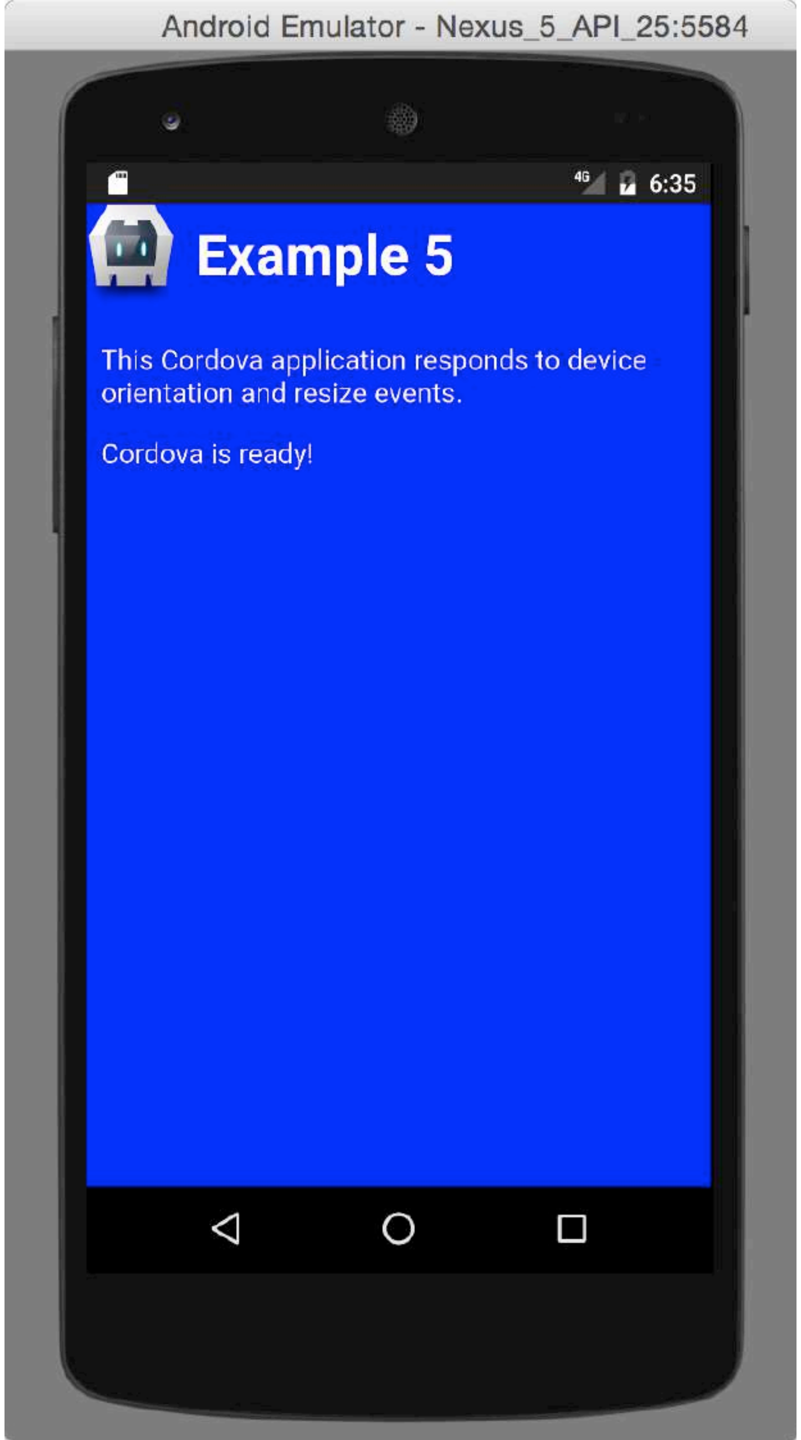
Orientation: 90degree
Screen width: 375
Screen height: 667
Window inner width: 667
Window inner height: 375



Example 5

This Cordova application responds to device orientation and resize events.

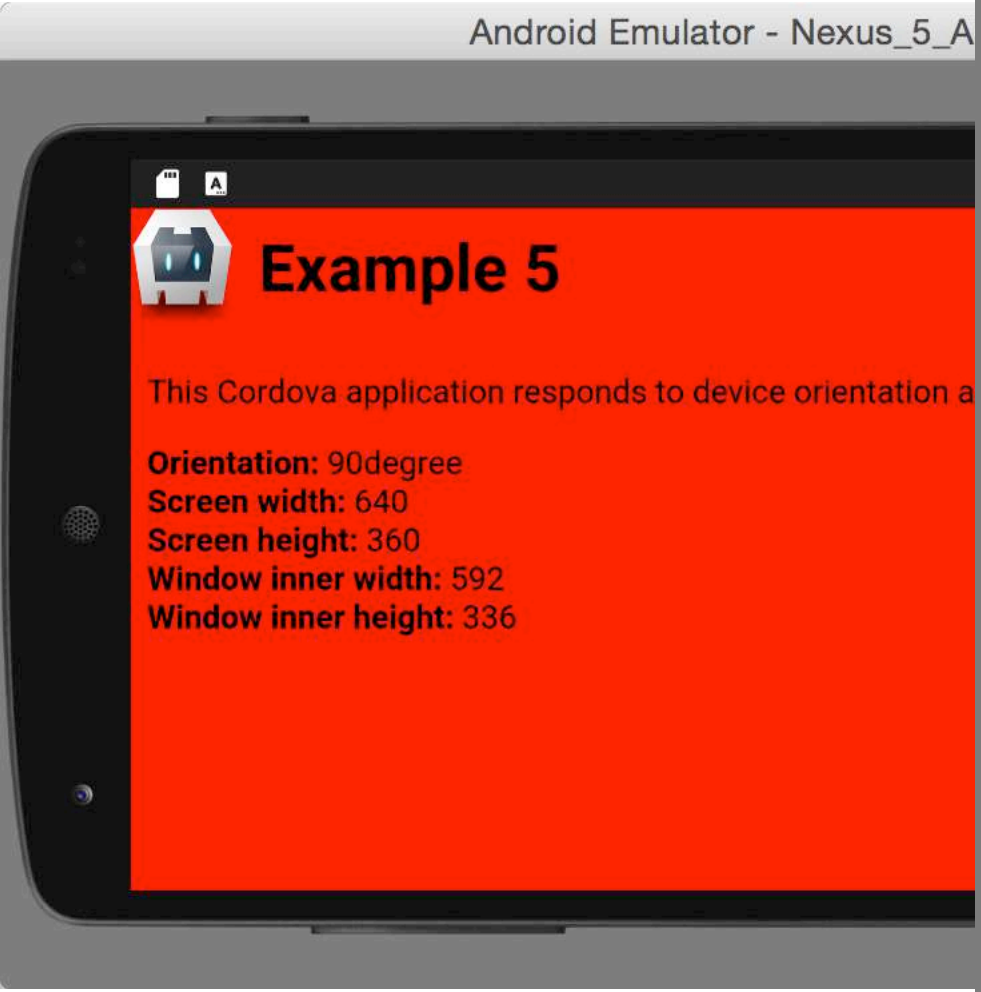
Orientation: 0degree
Screen width: 375
Screen height: 667
Window inner width: 375
Window inner height: 667



Example 5

This Cordova application responds to device orientation and resize events.

Cordova is ready!

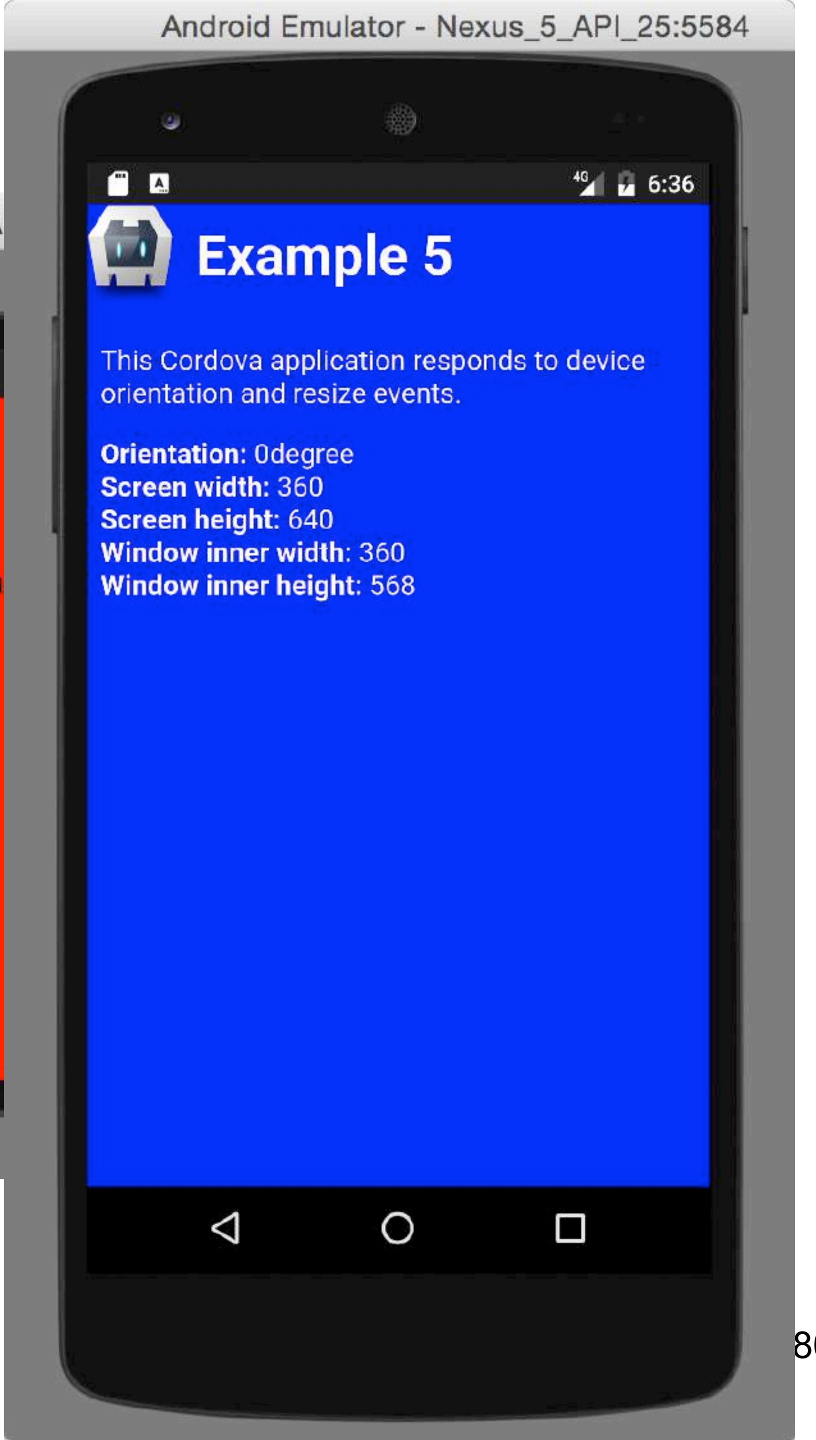


Android Emulator - Nexus_5_A

Example 5

This Cordova application responds to device orientation a

Orientation: 90degree
Screen width: 640
Screen height: 360
Window inner width: 592
Window inner height: 336



Android Emulator - Nexus_5_API_25:5584

6:36

Example 5

This Cordova application responds to device orientation and resize events.

Orientation: 0degree
Screen width: 360
Screen height: 640
Window inner width: 360
Window inner height: 568

Example 6

- Example 6 shows you how to get accelerometer readings from the device. A 3D accelerometer provides acceleration readings in three dimensions. In in each dimension, the reading ranges between 0 (stationary) and 10 (maximum).
- Support for access to accelerometer is through `cordova-plugin-device-motion`. The plugin provides object `accelerometer` in `navigator` namespace.
- The method to use is `getAcceleration` which requires two callbacks. One is to be called if acceleration readings obtained successfully, otherwise the other callback will be called.

```
navigator.accelerometer.getCurrentAcceleration(
        onSuccess, onError);
```

Graphical Interface

- The app displays a button “Refresh accelerometer reading”.

```
<button class="topcoat-button"
     onclick="app.getAccelerometer();">
     Refresh accelerometer reading
</button>
```

- The paragraph “appInfo” displays the current state of the app as well as the last accelerometer reading. Initially it displays “Connecting to device”.

```
<p id="appInfo"> Connecting to device ... </p>
```

- Once the `deviceready` event is received, it changes to “Device ready”.
- If the user clicks the button, a call to `app.getAccelerometer` is made and subsequently the results of the call is displayed in the appInfo area.
  - See `example06/www/index.html` for complete code.

JavaScript Code

(see `example06/www/js/index.js`)

```
var app = {

    initialize: function() {
        this.bindEvents();
        this.cordovaReady=false;
        this.appInfo=document.getElementById("appInfo");
        this.appInfo.innerHTML = "Conneting to device ...";
    },
    bindEvents: function() {
        document.addEventListener('deviceready',
               this.onDeviceReady, false);
    },
    onDeviceReady: function() {
        console.log("Device ready.");
        app.cordovaReady = true;
        app.appInfo.innerHTML="Device ready ...";
    },
```

JavaScript Code

```
 getAccelerometer: function() {
        console.log("Entered getAccelerometer()");
        if (this.cordovaReady) {
            navigator.accelerometer.getCurrentAcceleration(
                    this.onAccelSuccess,
                    this.onAccelFailure);
        } else {
            console.log("Cordova is not ready yet");
        }
    },
    onAccelSuccess(accel) {
        // update appInfo with accelerometer reading
    },
   onAccelFailure(errObj) {
        // report error in appInfo
   }
};

app.initialize();
```

onSuccess Callback

- If `getAcceleration` method successfully obtains the accelerometer reading, the onSuccess callback (`onAccelSuccess`) will be called.
- The callback will receive an object `accel` containing the current time `accel.timestamp` and acceleration reading along x, y, and z directions (`accel.x`, `accel.y`, and `accel.z`).

```
onAccelSuccess(accel) {
   var d = new Date(accel.timestamp);
   var s = '<ul class="topcoat-list__container">';
   s += '<li class="topcoat-list__header">Accelerometer Reading</li>';
   s += '<li class="topcoat-list__item"> X: '+accel.x+'</li>';
   s += '<li class="topcoat-list__item"> Y: '+accel.y+'</li>';
   s += '<li class="topcoat-list__item"> Z: '+accel.z+'</li>';
   s += '<li class="topcoat-list__item"> timestamp: ’
                +d.toLocaleString()+'</li>';
   s += '</ul>';
   app.appInfo.innerHTML = s;
},
```

onError Callback

- If `getAcceleration` method is unsuccessful in getting the accelerometer reading, the onError callback (`onAccelFailure`) will be called.
- The callback will receive an object `errObj` containing the error information

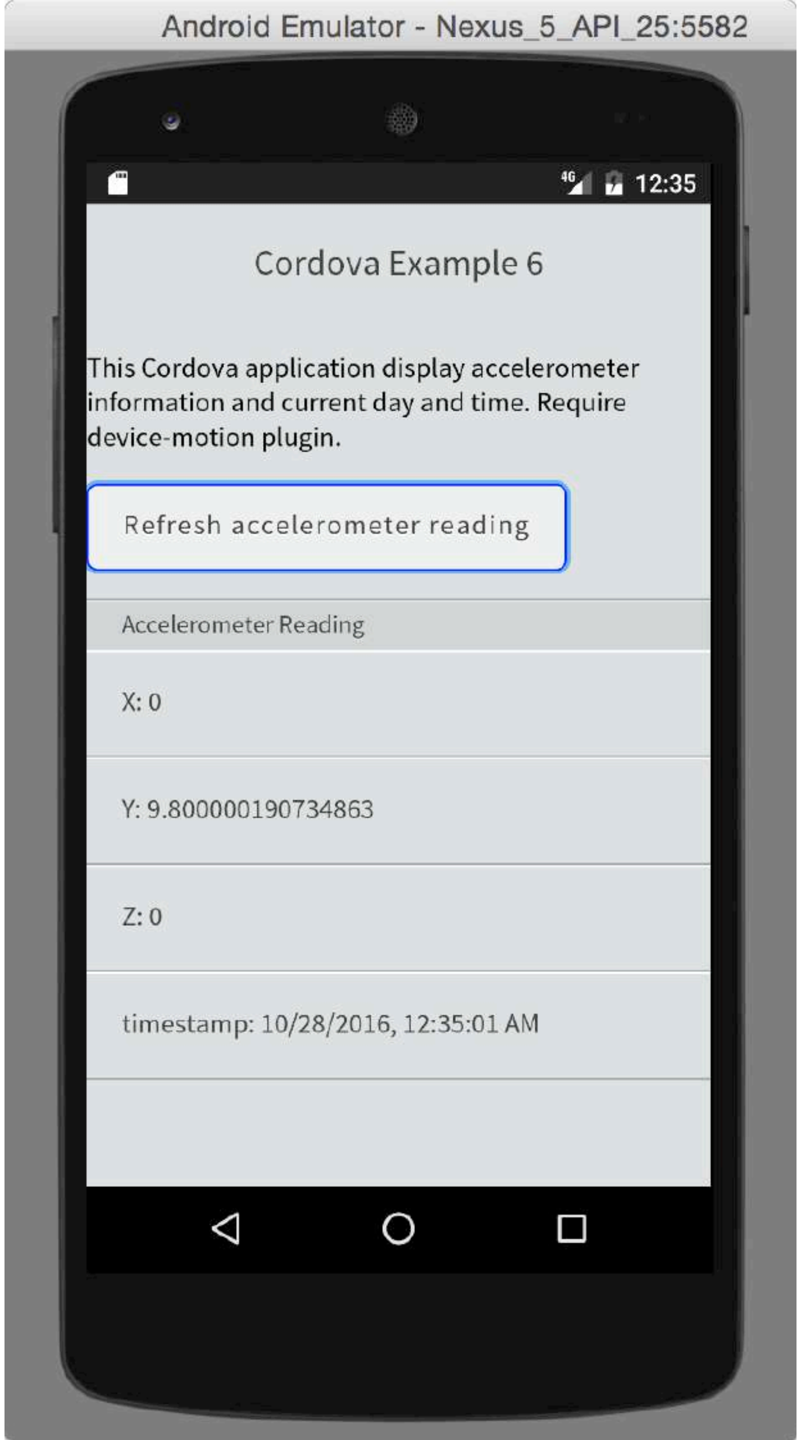
```
onAccelFailure(errObj) {
         console.log("Enter onAccelFailure");
         app.appInfo.innerHTML = JSON.stringify(errObj);
}
```

- In the above code, `JSON.stringify` method converts a JavaScript object into a string.

Other Notes

- Stylesheet:
  - In Example 6, we used a well-know CSS library from Adobe: topcoat.
  - The library is free, and you can download it from http://topcoat.io.

- Tests:
  - While most smartphones are now equipped with an accelerometer, support for accelerometer on browsers and device emulators vary.
  - I have tested the app on Google Chrome. It provides simulation of accelerometer.
  - Android emulator provides a simulated accelerometer, see the screenshot in the next slide.
  - However, there seems to be no support for accelerometer on the iPhone emulator, see the screen shot in the next slide.


Android Emulator - Nexus_5_API_25:5582

12:35

Cordova Example 6

This Cordova application display accelerometer information and current day and time. Require device-motion plugin.

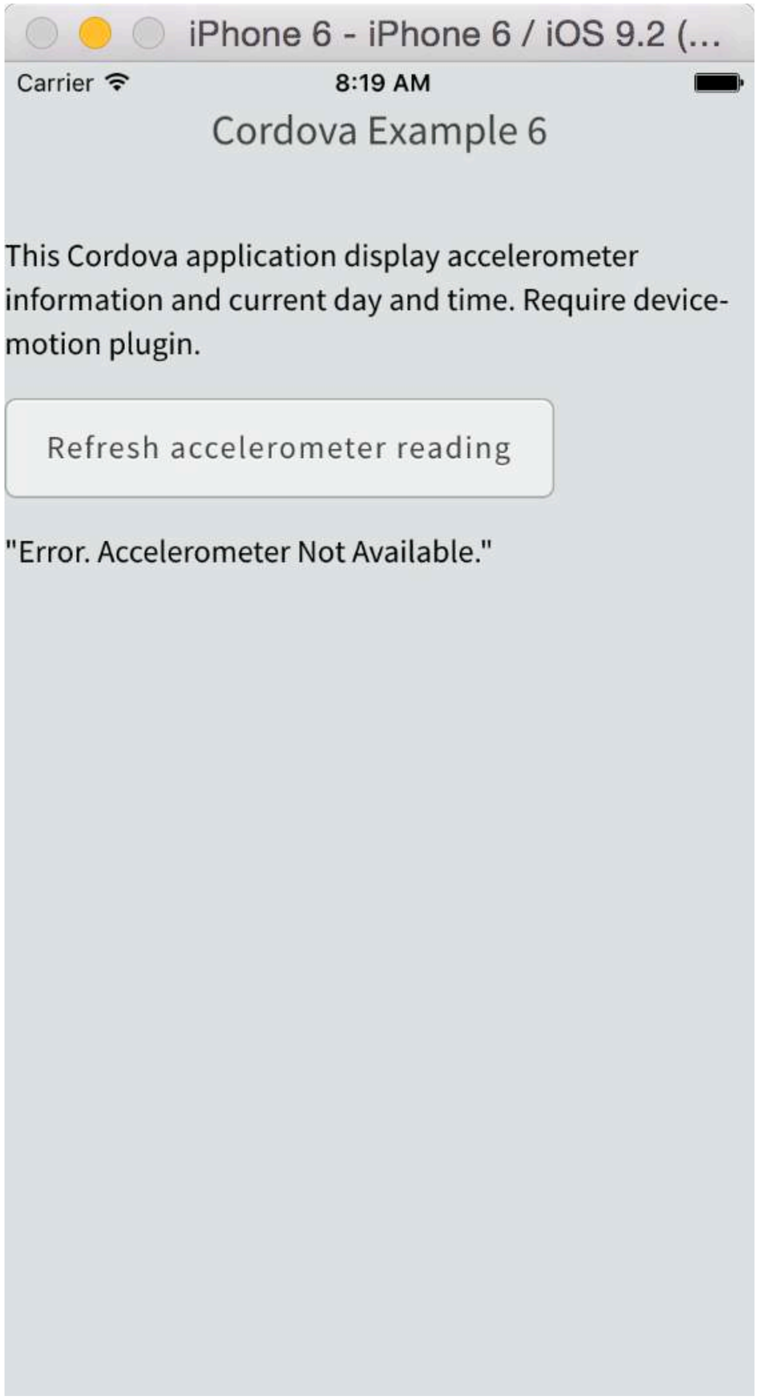
Accelerometer Reading

X: 0

Y: 9.800000190734863

Z: 0

timestamp: 10/28/2016, 12:35:01 AM


iPhone 6 - iPhone 6 / iOS 9.2 (...

Carrier 8:19 AM

Cordova Example 6

This Cordova application display accelerometer information and current day and time. Require device-motion plugin.

"Error. Accelerometer Not Available."

Example 7

- Example 7 shows you how to get the camera to take a picture.
- Support for camera is through `cordova-plugin-camera`. However the support on various devices and emulators are inconsistent and patchy at this moment.
- In this example, we will only show how to access the inbuilt camera application and instruct it to take a picture. However no attempt is made in this app to actually retrieve the picture that was taken. The later is more involved and sometimes requires a good deal of tweaking on different devices.
- The call to take a picture takes the form:

```
navigator.camera.getPicture(
        onSuccess, onError, options);
```

Graphical Interface

- The app displays a button “Take a picture”.

```
<button class="topcoat-button”
     onclick="app.takePicture();">
     Take a picture
</button>
```

- The paragraph “appInfo” displays the current state of the app as well as the last accelerometer reading. Initially it displays “Connecting to device”.

```
<p id="appInfo"> Connecting to device ... </p>
```

- Once the `deviceready` event is received, it changes to “Device ready”.
- If the user clicks the button, a call to `app.takePicture` is made to access the inbuilt camera application.
  - See `example07/www/index.html` for complete code.

JavaScript Code

(see `example07/www/js/index.js`)

```
var app = {
    initialize: function() {
        this.bindEvents();
        this.deviceReady=false;
        this.ai=document.getElementById("appInfo");
        this.ai.innerHTML = "Conneting to device ...";
    },
    bindEvents: function() {
        document.addEventListener('deviceready',
               this.onDeviceReady, false);
    },
    onDeviceReady: function() {
        console.log("Entered onDeviceReady.");
        app.deviceReady = true;
        app.ai.innerHTML="Device ready ...";
    },
```

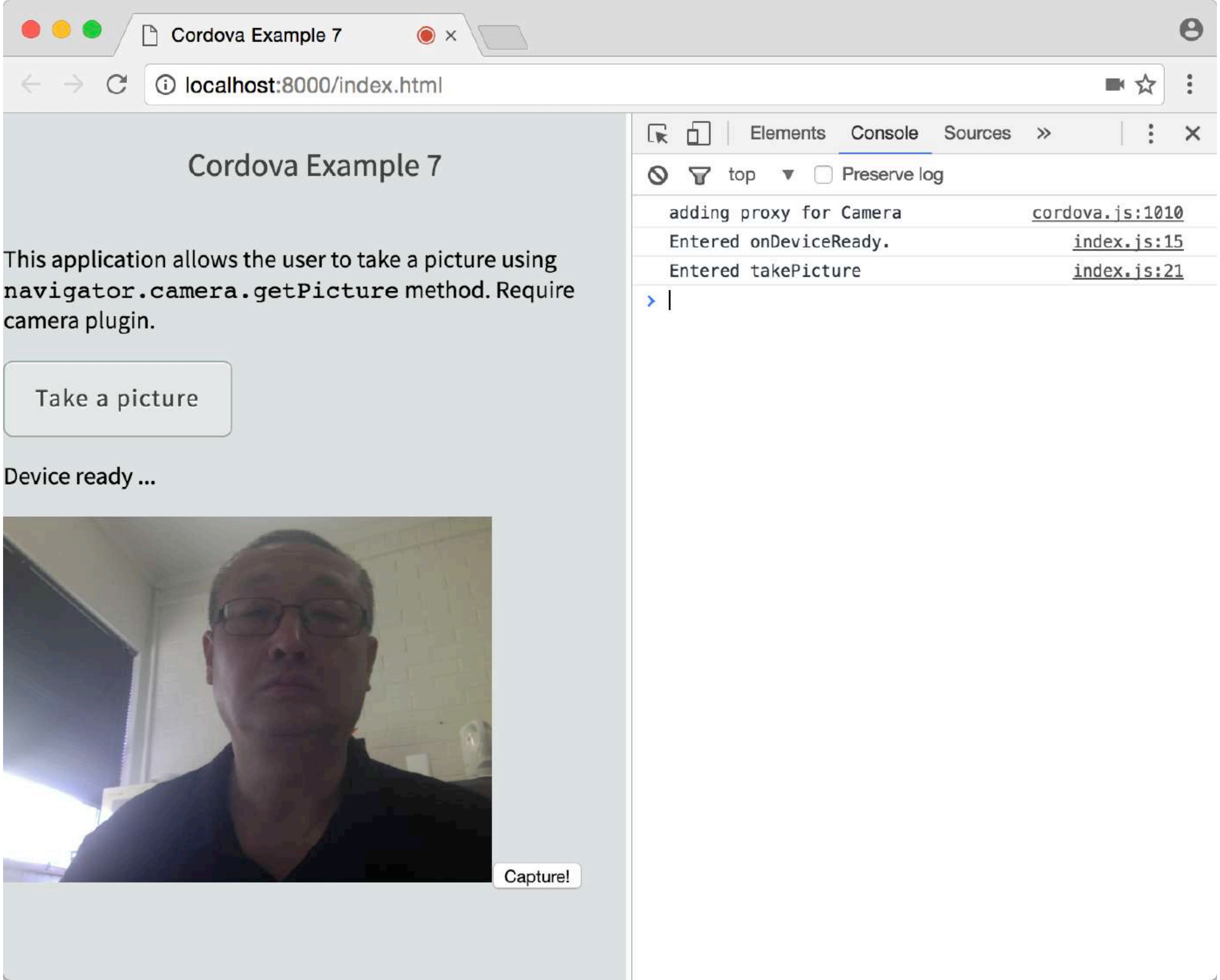
JavaScript Code

```
   takePicture: function() {
       console.log("Entered takePicture");
       if (this.deviceReady) {
           navigator.camera.getPicture(
               this.onCameraSuccess, this.onCameraError,
               {destinationType: Camera.DestinationType.File_URI});
       }
   },
   onCameraSuccess: function(imageURI) {
       console.log("Enter onCameraSuccess");
       app.ai.innerHTML = "Image is in: "+JSON.stringify(errObj);
   },
   onCameraError(message) {
       console.log("Enter onCameraError");
       app.ai.innerHTML = message;
   }
};

app.initialize();
```

navigator.camera.getPicture

- The method getPicture requires three arguments.
  - The first one is a callback (`onCameraSuccess`) if the method has successfully accessed inbuilt camera app.
  - The second one is a callback (`onCameraError`) if the method fails to gain access to the inbuilt camera app.
  - The last one is an object contains various options. For details see the following Cordova documentation:
    - https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera/index.html#module_camera.getPicture

- Tests show that
  - Google Chrome has access to the computer's camera.
  - Android emulator supports the camera operation.
  - iPhone emulator doesn't support camera
  - Test on an android phone (Mi Note LTE, Android 6.01) shows that app can access the camera app on the device.

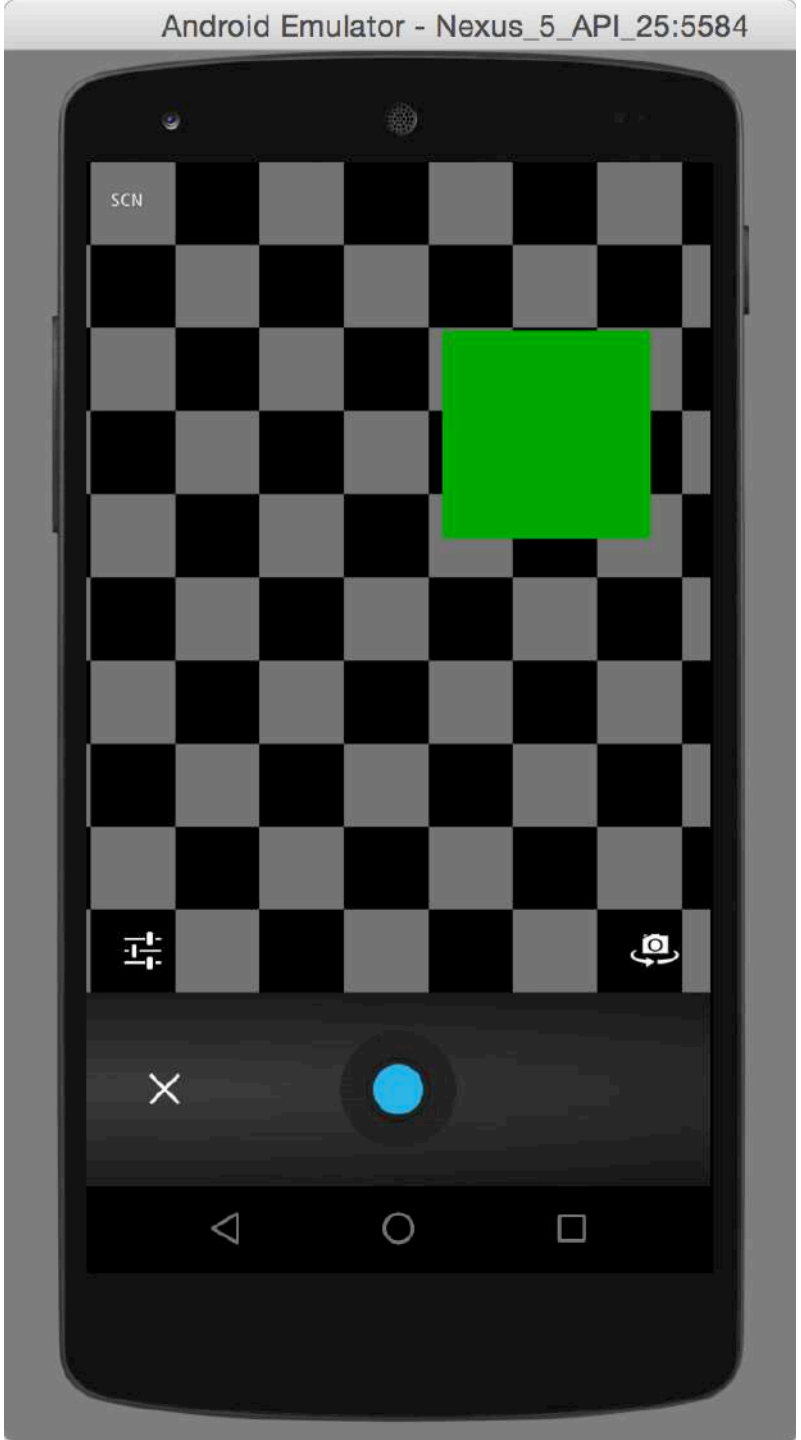


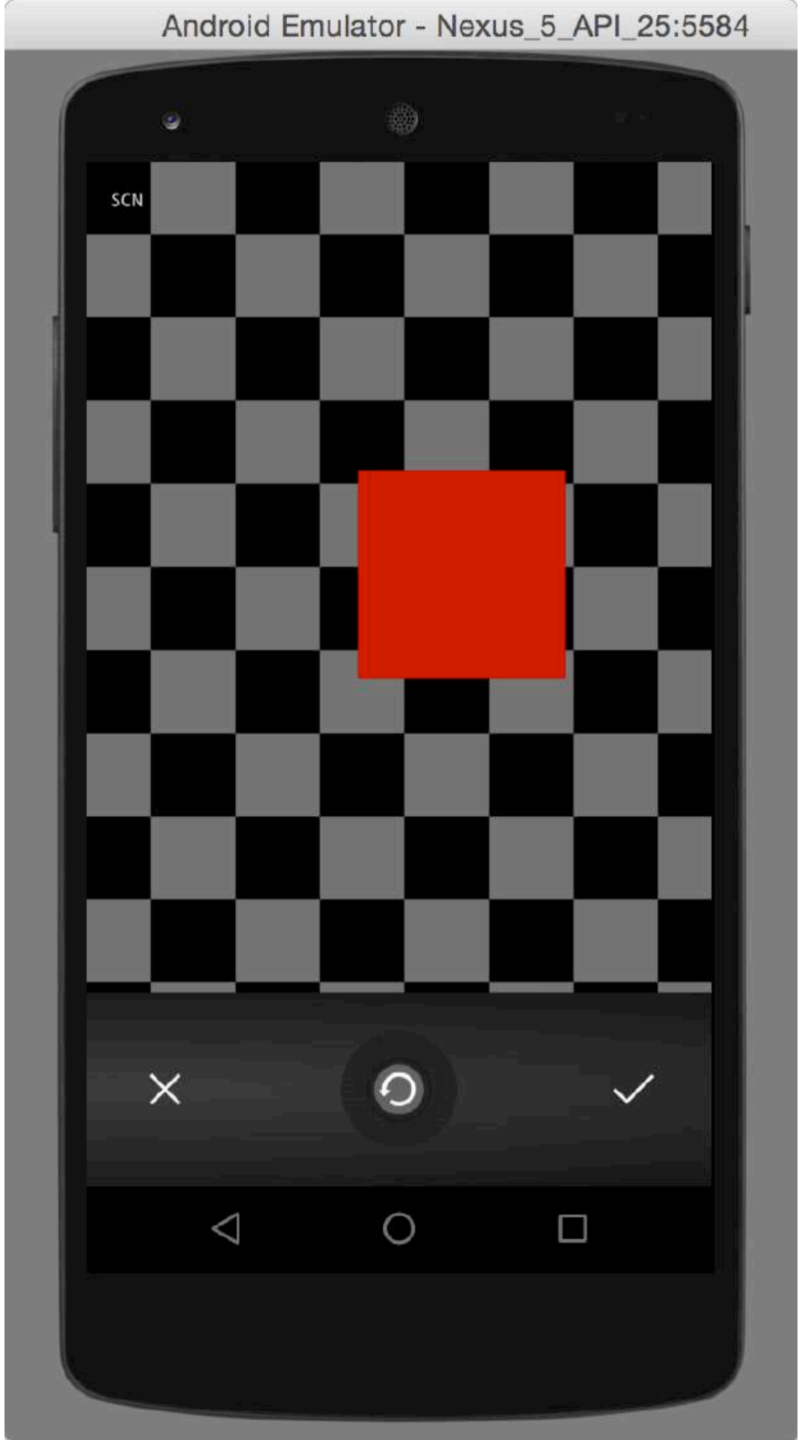
Cordova Example 7

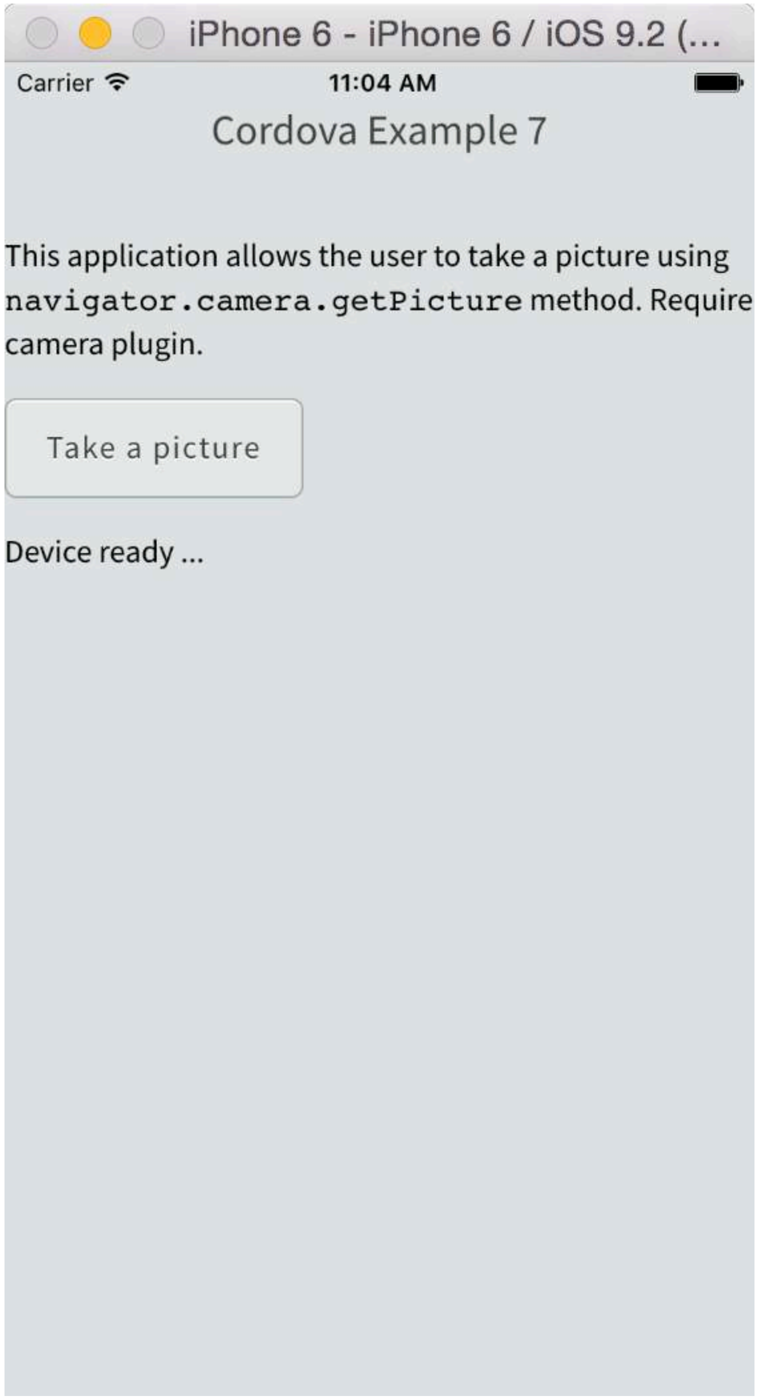
This application allows the user to take a picture using `navigator.camera.getPicture` method. Require camera plugin.

Device ready ...

```
adding proxy for Camera          cordova.js:1010
Entered onDeviceReady.           index.js:15
Entered takePicture              index.js:21
```


Android Emulator - Nexus_5_API_25:5584
SCN


Android Emulator - Nexus_5_API_25:5584
SCN

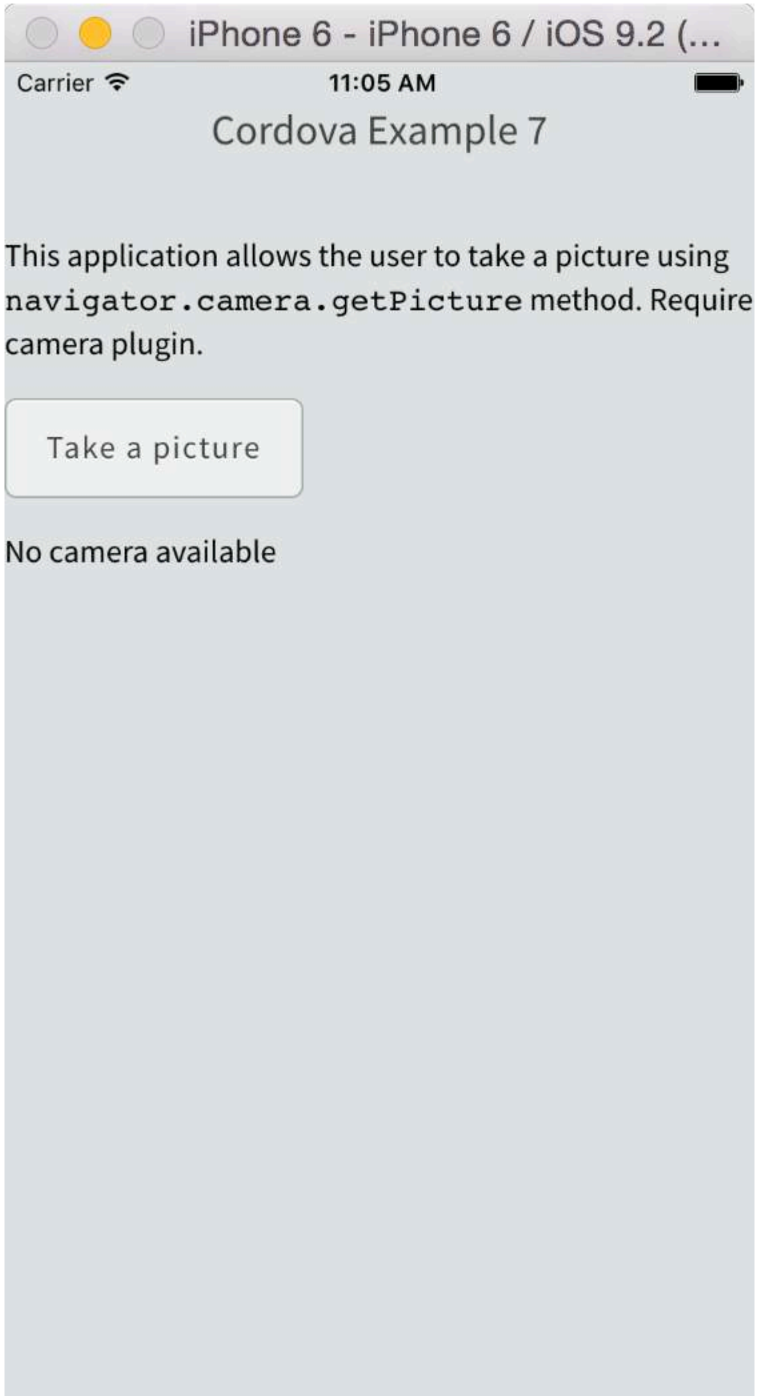


Cordova Example 7

This application allows the user to take a picture using `navigator.camera.getPicture` method. Require camera plugin.

Take a picture

Device ready ...

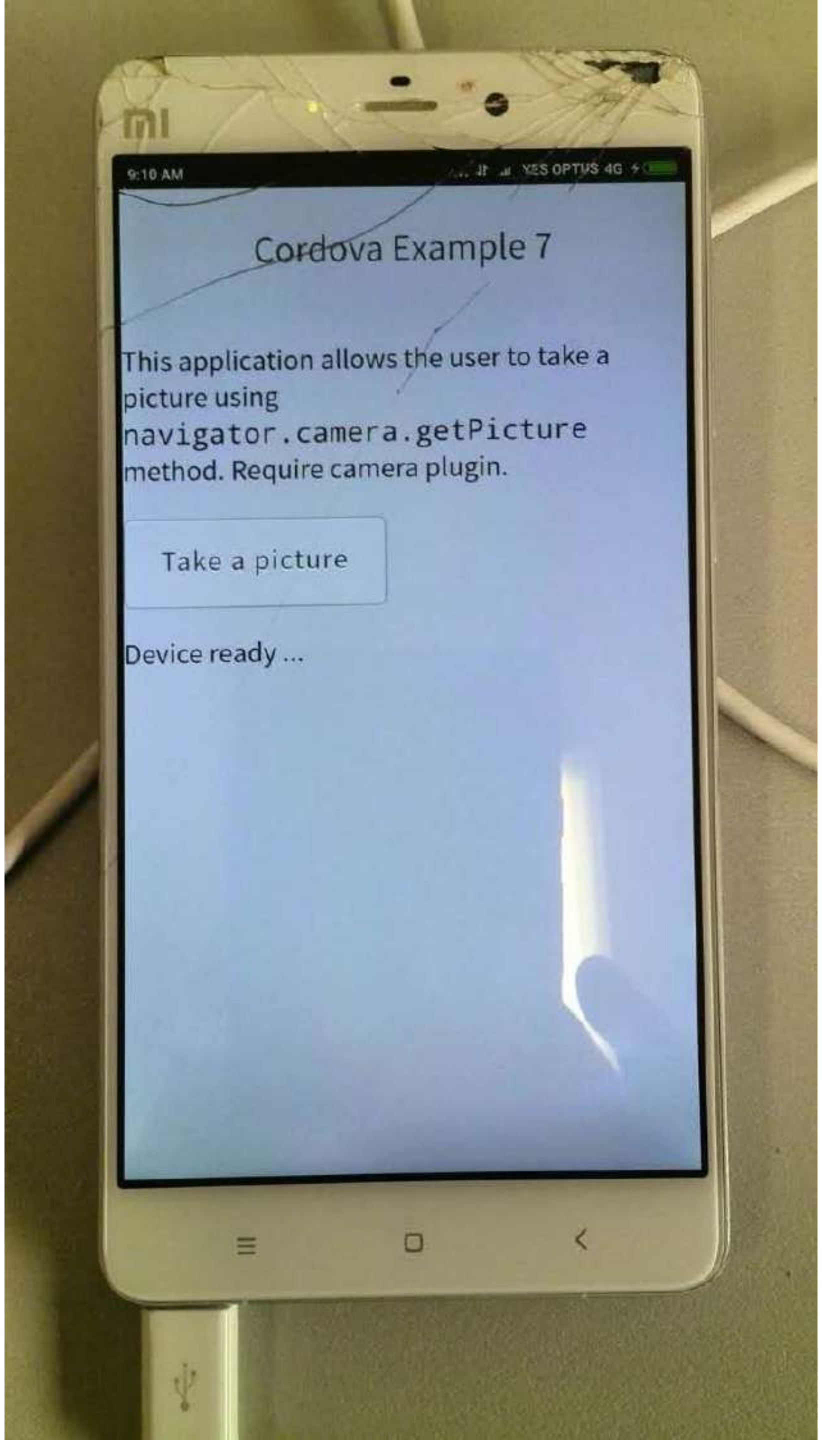


Cordova Example 7

This application allows the user to take a picture using `navigator.camera.getPicture` method. Require camera plugin.

Take a picture

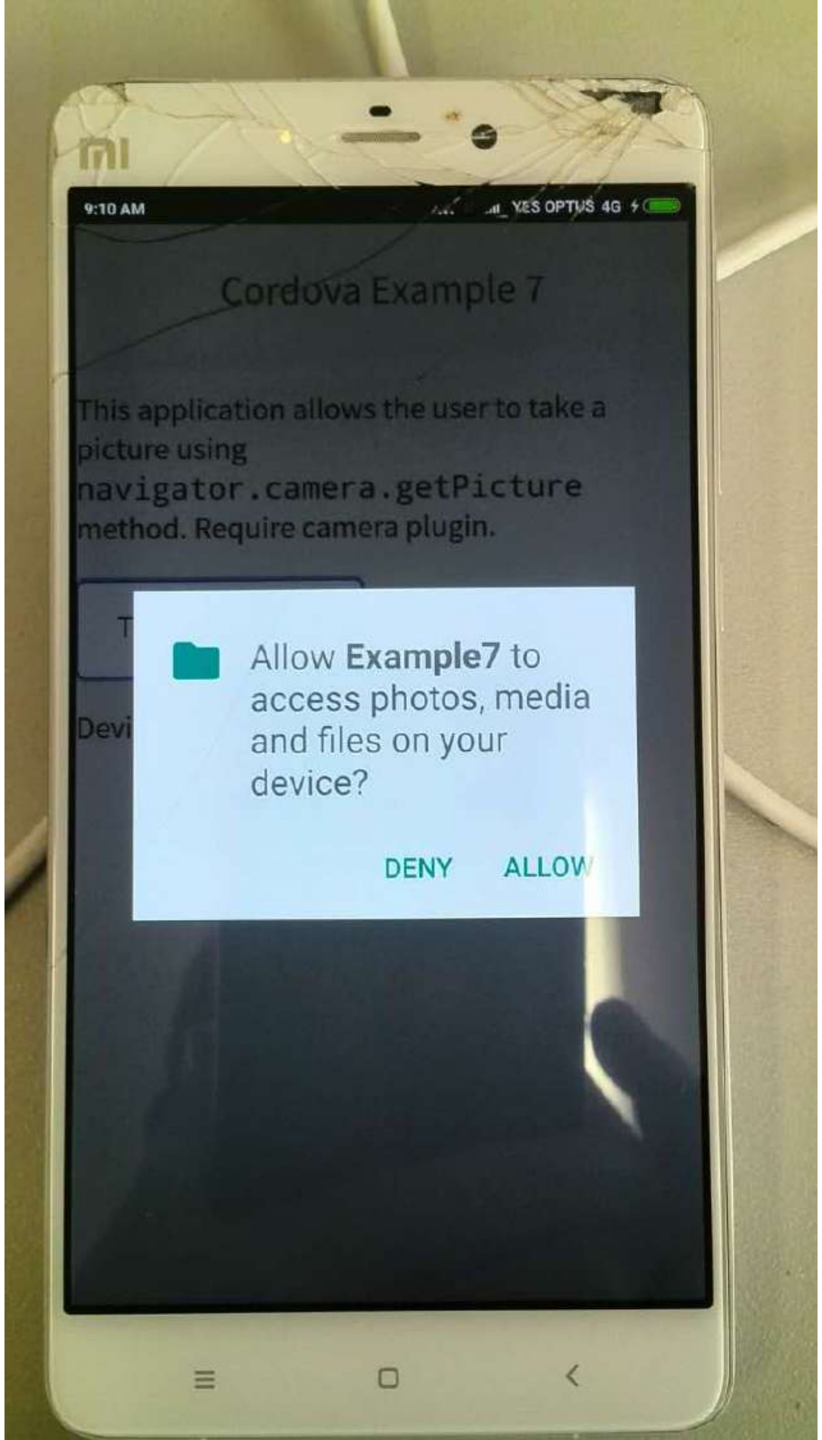
No camera available



Cordova Example 7

This application allows the user to take a picture using `navigator.camera.getPicture` method. Require camera plugin.

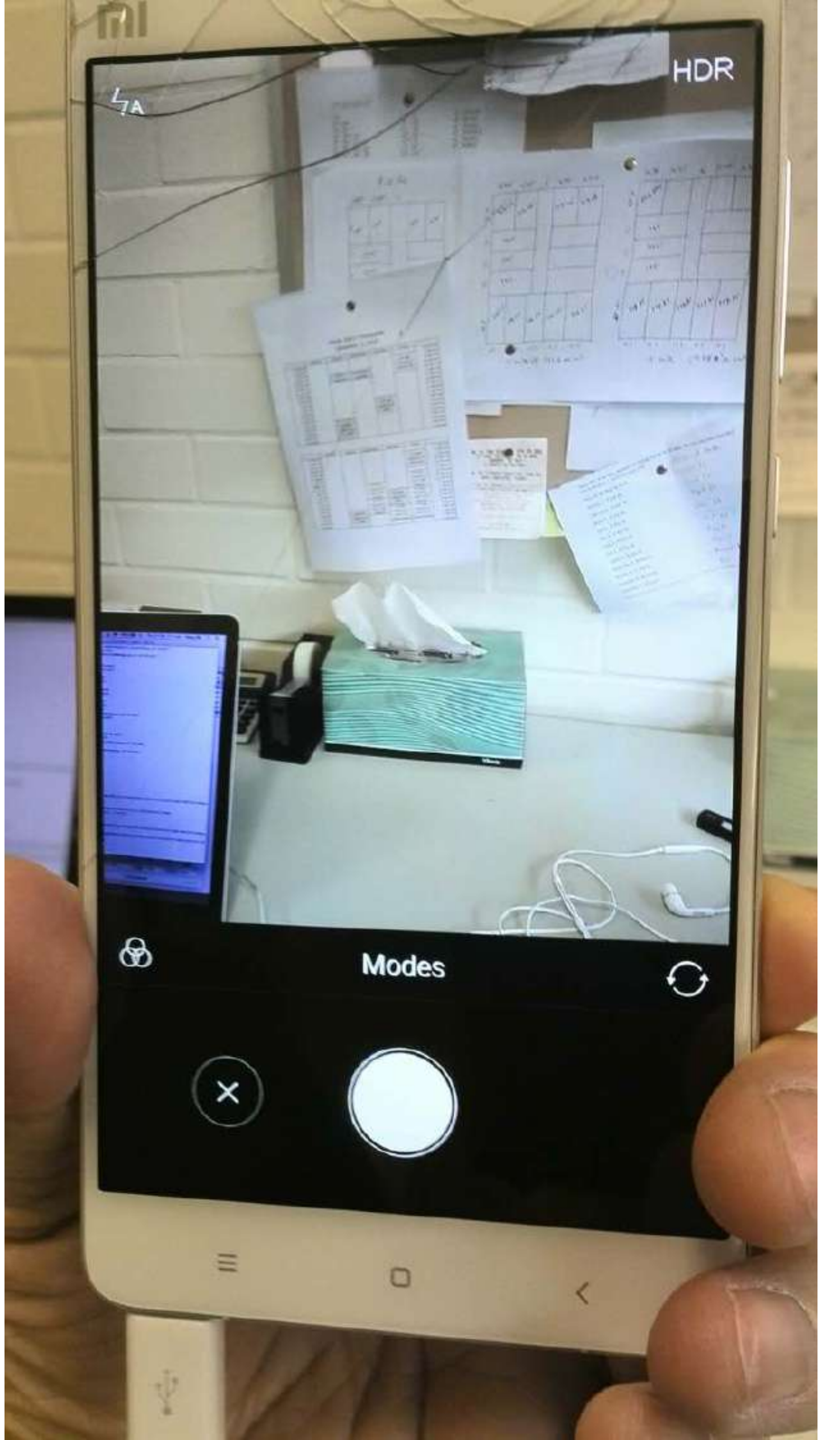
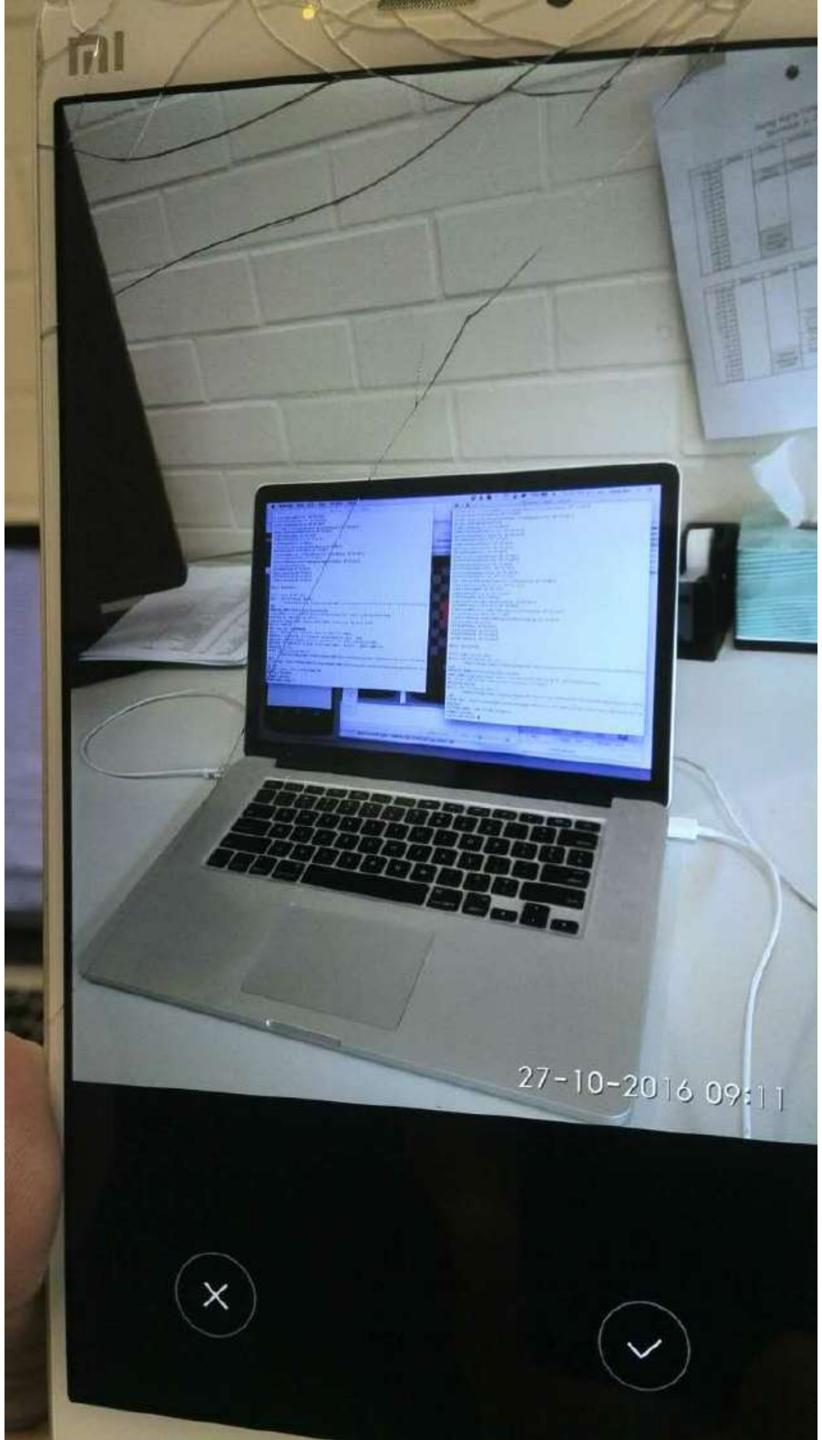
Device ready ...



Cordova Example 7

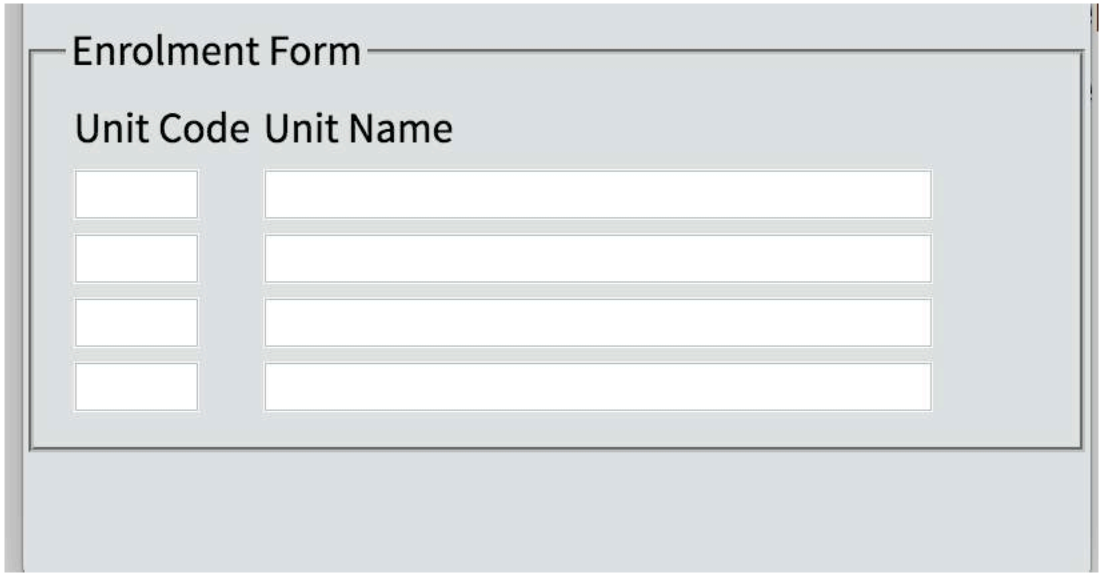
This application allows the user to take a picture using `navigator.camera.getPicture` method. Require camera plugin.

Allow **Example7** to access photos, media and files on your device?


HDR
Modes

27-10-2016 09:11

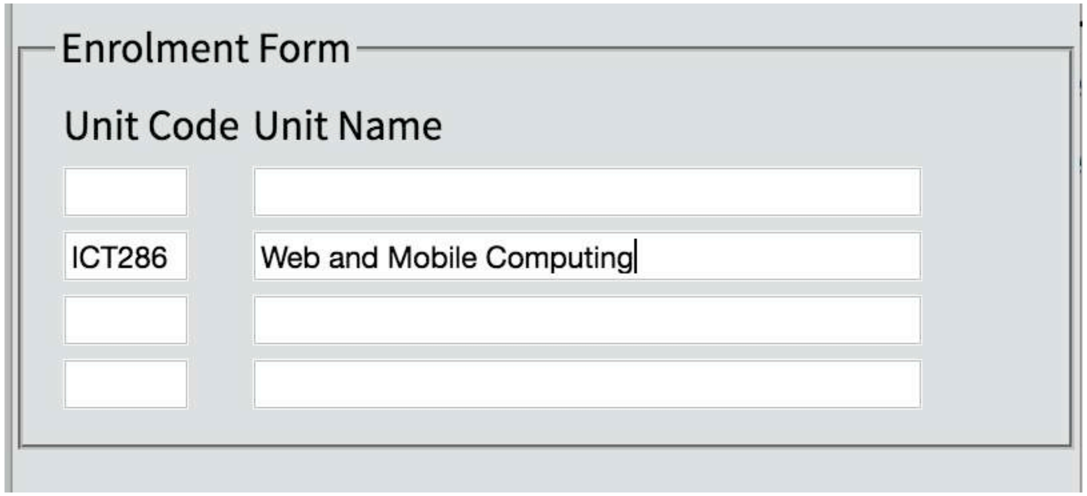
Example 8

- The last example is a client-server example. The client displays an HTML form inviting the user to enter the unit information including the unit code and unit name. Each textbox for a unit code monitors the blur event, which will trigger a call to a JavaScript method to retrieve the unit name corresponding to the unit code from the server.



Retrieve Unit Name from the Server

- This JavaScript method, `getUnitName`, uses Ajax to send an HTTP request to the server script `getUnitName.php` and once the response comes back from the server script, it enters the unit name in the unit name textbox for that unit code, saving the user from typing the long unit name into the textbox.


Enrolment Form

Unit Code	Unit Name
ICT286	Web and Mobile Computing

getUnitName Method

- See `www/js/index.js` for complete code

```
 getUnitName: function(unitCode, id) {
     console.log(unitCode + " " + id);
     var xhr = new XMLHttpRequest();
     xhr.onreadystatechange = function() {
         if (xhr.readyState == 4 && xhr.status == 200)
             if (xhr.responseText) {
                 console.log("responseText: " + xhr.responseText);
                 document.getElementById(id).value
                      = xhr.responseText;
             }
     }
     xhr.open("GET",
"http://ceto.murdoch.edu.au/~s900432d/ICT286/getUnitName.php?UnitCode=
" + unitCode);
     xhr.send(null);
 },
```

Note: the above code uses Ajax to retrieve the unit name. We will discuss Ajax in a later topic. For now, just accept it if you don't understand it.

getUnitName.php

- See `server/getUnitName.php` for complete code

```php
<?php
   header("Access-Control-Allow-Origin: http://localhost:8000");
   $units = array(
        "ICT286" => "Web and Mobile Computing",
        "ICT374" => "Operating Systems and Systems Programming",
        "ICT397" => "Advanced Games Design and Programming",
        "ICT375" => "Advanced Web Programming",
        "ICT365" => "Software Development Frameworks",
        "ICT517" => "Advanced IT Project",
        "ICT167" => "Principles of Computer Science"
        );
   $unitCode = $_GET["UnitCode"];
   if (array_key_exists($unitCode, $units))
        print $units[$unitCode];
   else
        print "";
?>
```

Note: we will discuss PHP programming in the next topic.

Cross-Origin Response

- Many web browsers do not accept the response from a cross-origin server. Eg, in our `getUnitName` method, we use `xhr` to create and send an HTTP request to the server:

```
 xhr.open("GET",
"http://ceto.murdoch.edu.au/~s900432d/ICT286/getUnitName.ph
p?UnitCode=" + unitCode);
 xhr.send(null);
```

  - The request comes from http://localhost:8000/
  - But the response comes from http://ceto.murdoch.edu.au/
- This response is a cross-origin response. For security reason, many browser would reject such a response.
- In order for your browser to accept a cross-origin response, you need to add `Access-Control-Allow-Origin` header into your response:

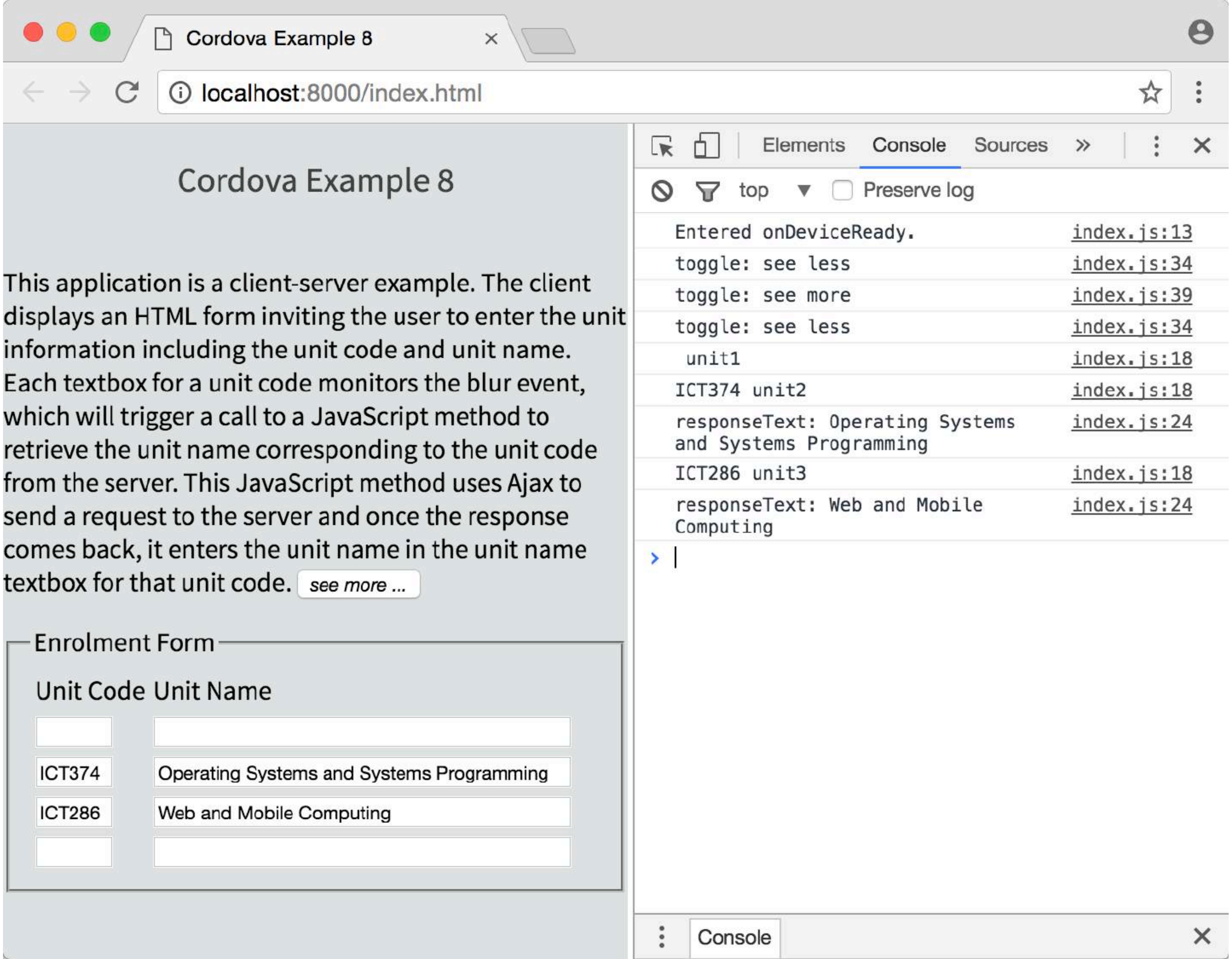
```
  header("Access-Control-Allow-Origin:
http://localhost:8000");
```

Configure Your Mobile App

- You can configure your mobile app so that it can only access a particular location in the Internet.
- In this example, you add the following element in the configuration file `config.xml`:

```
<access origin="http://ceto.murdoch.edu.au/*" />
```

  - This means that our mobile app can only get response from the ceto server.
  - Note that the default configuration allows the app to access any place.



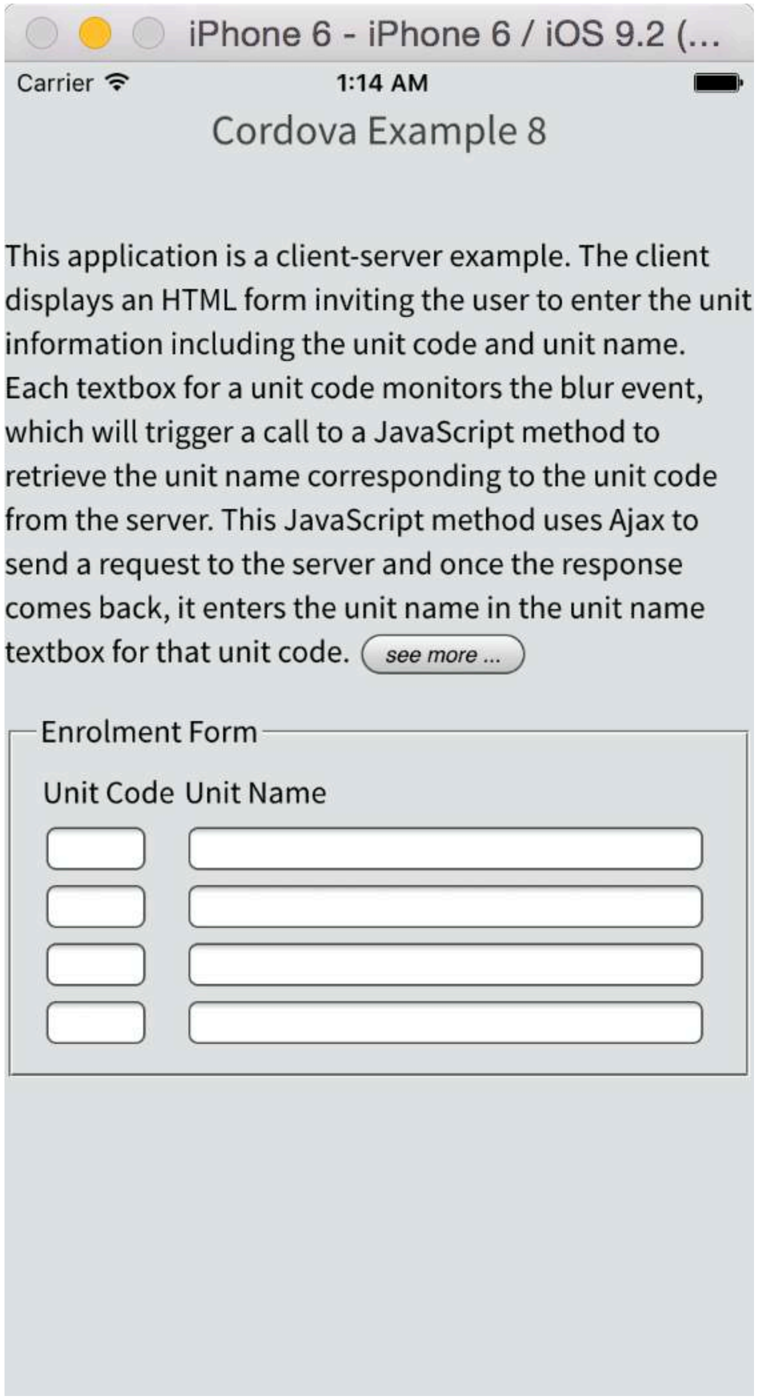
Cordova Example 8

This application is a client-server example. The client displays an HTML form inviting the user to enter the unit information including the unit code and unit name. Each textbox for a unit code monitors the blur event, which will trigger a call to a JavaScript method to retrieve the unit name corresponding to the unit code from the server. This JavaScript method uses Ajax to send a request to the server and once the response comes back, it enters the unit name in the unit name textbox for that unit code. see more ...

Enrolment Form

Unit Code	Unit Name
ICT374	Operating Systems and Systems Programming
ICT286	Web and Mobile Computing

```
Entered onDeviceReady.                      index.js:13
toggle: see less                            index.js:34
toggle: see more                            index.js:39
toggle: see less                            index.js:34
 unit1                                      index.js:18
ICT374 unit2                                index.js:18
responseText: Operating Systems             index.js:24
and Systems Programming
ICT286 unit3                                index.js:18
responseText: Web and Mobile                index.js:24
Computing
>
```

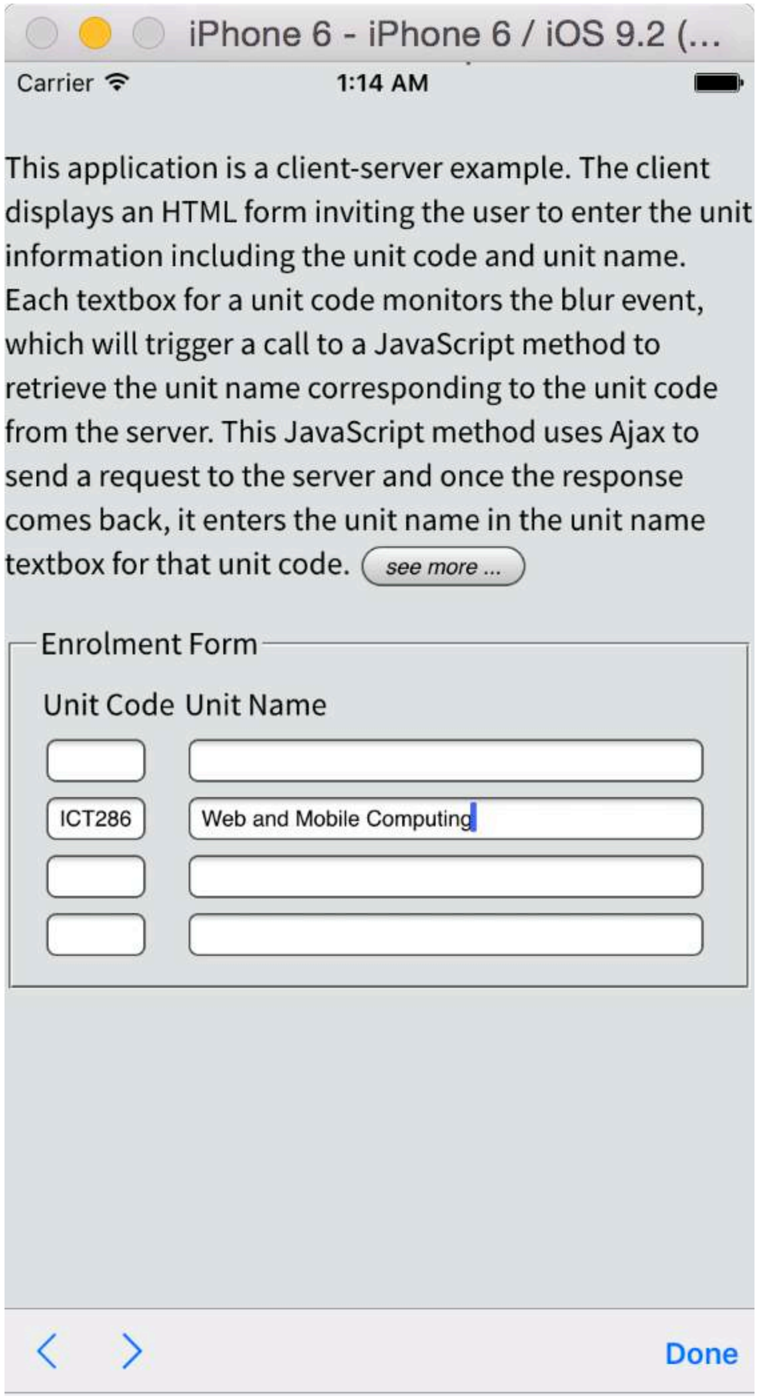


Cordova Example 8

This application is a client-server example. The client displays an HTML form inviting the user to enter the unit information including the unit code and unit name. Each textbox for a unit code monitors the blur event, which will trigger a call to a JavaScript method to retrieve the unit name corresponding to the unit code from the server. This JavaScript method uses Ajax to send a request to the server and once the response comes back, it enters the unit name in the unit name textbox for that unit code. see more ...

Enrolment Form

Unit Code	Unit Name



This application is a client-server example. The client displays an HTML form inviting the user to enter the unit information including the unit code and unit name. Each textbox for a unit code monitors the blur event, which will trigger a call to a JavaScript method to retrieve the unit name corresponding to the unit code from the server. This JavaScript method uses Ajax to send a request to the server and once the response comes back, it enters the unit name in the unit name textbox for that unit code. see more ...

Enrolment Form

Unit Code	Unit Name
ICT286	Web and Mobile Computing

Further Work

- The best and most up to date information is in Cordova Documentation:

  https://cordova.apache.org/docs/en/latest/

- Suggestions
  - To learn more about Cordova, you need practice.
  - It is essential that you install Cordova on your own computer in order to do more practice. Follow the instructions to install Cordova on your own computer – this include
    - Installing Android Studio and SDK
    - Installing Xcode (if you use a Mac)
    - Installing Node.js
    - Installing Git
    - Installing Cordova