Using Visual Studio Code to build iOS and Android apps using Apache Cordova

# Overview

Apache Cordova is an open-source framework that allows you to write apps for Android, iOS, Windows, and other mobile platforms, using nothing but HTML, CSS, and JavaScript. The apps built by Cordova are called hybrid apps - they aren't native apps and they aren't simple web apps, but a combination of both. Hybrid apps can have access to native device APIs and their layout rendering is done using Web views instead of the platform's native UI framework.

To help you build apps using Cordova, the Microsoft Visual Studio team has created a Cordova Extension for Visual Studio Code. These tools help setup your dev environment quickly as well as offer a consistent editing and debugging experience across all your target mobile devices.

This tutorial takes you through the basics of Apache Cordova, by building a simple Getting Started app using Visual Studio Code and deploying it to an iOS simulator or Android device.

**Note:** For this code challenge, you’ll be using either an iOS simulator or Android device, be sure to follow the sections that are a fit for your workstation setup.

# Objectives

* Learn how to create a new Apache Cordova project in Visual Studio Code using the Cordova Extension.
* Understand the basics of the Ionic framework, a popular JavaScript framework for Cordova developers.
* See how you can use Visual Studio Code to debug and test your app on Android and iOS devices.

# Prerequisites

The following has already been setup for this code challenge:

* Visual Studio Code: <http://code.visualstudio.com>
* Cordova Extension for VS Code: <http://aka.ms/VSCodeCordovaExtension>
* For iOS development on a Mac:
  + Homebrew: <http://brew.sh>
  + Open a terminal and run:

$ brew install ideviceinstaller ios-webkit-debug-proxy

* Cordova and Ionic, installed via NPM:

$ npm install -g cordova ionic

* Android SDK for Android Development: By following the Cordova Android Platform Guide: <http://aka.ms/CordovaAndroidPlatformGuide>
* XCode and XCode Command Line Tools for iOS development: XCode installed from the App Store, Command line tools installed from the Terminal:

$ xcode-select --install

# This quick start is for you, if…

This Quick Start Challenge is for any developer already familiar with web development.

# Step 1: Create a new project from the Cordova command line

Let’s start by creating a new project using the Ionic command line interface (CLI).

**Note**: This challenge uses the Ionic Framework, which is a popular JavaScript/CSS framework built for Cordova development. You can learn more about it at <http://www.ionicframework.com>.

First, **open a new instance of the Terminal (on a Mac) or Command Prompt (on Windows).**

**Quick-Tip:** On a Mac you can get to the Terminal by pressing **Command+Space Bar** on the keyboard and typing **Terminal**. On Windows 10, type **Windows key+S** and type **Cmd**.

Type the following command to create a new project using the Ionic tabs template:

ionic start myApp tabs

If the myApp directory already exists, overwrite it.

As the app is being created, you’ll be prompted with the following question. Respond with “n” and hit enter, this challenge will not be using Push Notifications:

Create an ionic.io account to send Push Notifications and use the Ionic View app?

(Y/n): n

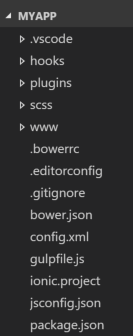
Change into the directory for this project:

cd myApp

Let’s launch the Visual Studio Code editor now, to view the contents of this project. Type the following and hit enter to launch the editor:

code .

You'll now see the default project structure in your solution explorer, it should look like the following.



A few things worth calling out here:

* The www folder is where your main app source code lives. In this template, www/index.html is the first screen of your app.
* config.xml is the Cordova configuration file.
* The plugins folder is a home for plugins. In Cordova, you use plugins to access native device capabilities, like the camera.

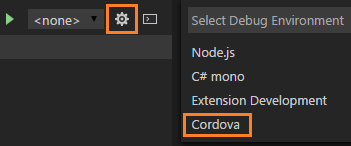
# Step 2a: Run the app on an Android device

**Note:** if you’re on a Mac without an Android device at your station, skip this section.

**Switch back to the Terminal (on Mac) or Command Prompt (on Windows)** and add Android platform support for your app by typing the following:

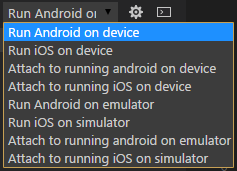
ionic platform add android

Let's run the app on an Android device and see what it looks like. In Visual Studio Code, Click the **debug icon** (Choose Cordova debugger) in the **View bar**, and then click the **configure gear** icon (Configure-gear) to choose the **Cordova** debug environment.



A **launch.json** file will open, which contains default debugging configurations. For this challenge, you can **close the launch.json file**.

To start the debugger, choose the **Run Android on Device** option from the **target drop-down list**, and then click the start button (Configure-gear).



**Note:** This is where you can go to select iOS platform targets, devices, and emulators as well.

Once the build is completed, you’ll see the app open up on your screen.

# 

# Step 2b: Run the app on the iOS simulator

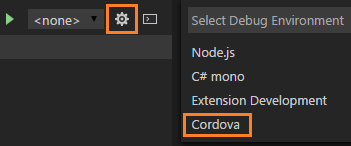
**Note:** if you’re usingan Android device at your station, skip this section.

Let's run the app on an iOS simulator and see what it looks like.

**Switch back to the Terminal (on Mac) or Command Prompt (on Windows)** and add iOS platform support for your app by typing the following:

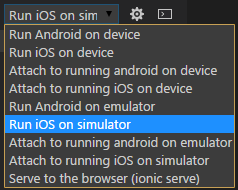
ionic platform add ios

In Visual Studio Code, **click the debug icon** (Choose Cordova debugger) in the **View bar**, and then click the **configure gear** icon (Configure-gear) to choose the **Cordova** debug environment.



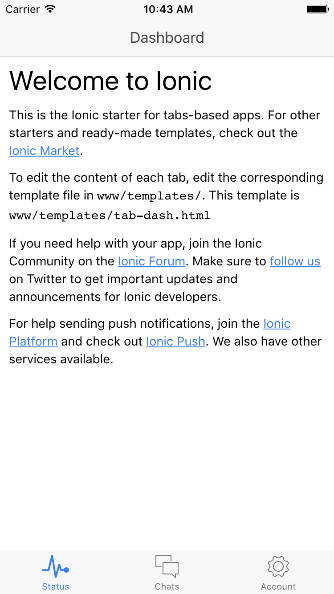
A **launch.json** file will open, which contains default debugging configurations. For this challenge, you can **close the launch.json file**.

To start the debugger, choose the **Run iOS on simulator** option from the **target drop-down list**, and then click the start button (Configure-gear).



**Note:** This is where you can go to select Android platform targets, devices, and emulators as well.

Once the build is completed, you’ll see the app open up on your screen.



# Step 3: Access native device capabilities

With Cordova, you can get at native device capabilities (such as the Camera) by using Plugins. Plugins provide a common JavaScript API that you use across device platforms, and the plugin author has done the work to implement the native functionality for each platform.

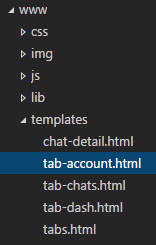
Let’s add a button to our application that will setup your account photo, which could be used when you send messages from within the app. To do this, you’ll be using the Camera plugin.

**Switch to the Terminal (on Mac) or Command Prompt (on Windows)** and add the camera plugin to your project

$ ionic plugin add cordova-plugin-camera

Once the plugin has installed, go ahead and close this file.

Now, let’s add a button to this app that will open the camera when clicked and then, after a picture is taken, show it in the UI. In the project, open the **www\templates\tab-account.html** file.



This file defines the UI you see when clicking on the **Account tab** in the running application. When the file opens, you’ll see the following source:

<ion-view view-title="Account">

<ion-content>

<ion-list>

<ion-toggle ng-model="settings.enableFriends">

Enable Friends

</ion-toggle>

</ion-list>

</ion-content>

</ion-view>

Change it to include the following code that will add a button and a placeholder image to show the picture you’ll take from the camera.

<ion-view view-title="Account">

<ion-content>

<ion-list>

<ion-toggle ng-model="settings.enableFriends">

Enable Friends

</ion-toggle>

<ion-item>

<button class="button" ng-click="takePicture()">Set My Picture</button>

<img ng-src="{{imageSource}}" width="100" height="100">

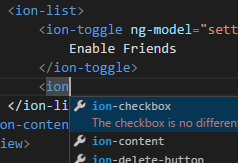
</ion-item>

</ion-list>

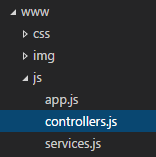
</ion-content>

</ion-view>

Notice, as you type there are IntelliSense suggestions provided for the Ionic-specific elements:



Now to add the functionality behind the button, open the **www\js\controllers.js** file.



This file defines a set of controllers, using the Angular JS library. These controllers are responsible for coordinating actions between the backend data model and the UI components. In this file, you’ll find the following code that defines the behavior for the Account tab:

.controller('AccountCtrl', function ($scope) {

$scope.settings = {

enableFriends: true

};

});

Add this last snippet of code, which defines a takePicture() function that is called by the button on the Account tab.

.controller('AccountCtrl', function ($scope) {

$scope.settings = {

enableFriends: true

};

$scope.takePicture = function () {

navigator.camera.getPicture(function (imageData) {

$scope.$apply(function () {

$scope.imageSource = "data:image/jpeg;base64," + imageData;

});

}, function (err) { }, {destinationType: Camera.DestinationType.DATA\_URL});

}

});

Breaking this down, here’s a summary of what this code does:

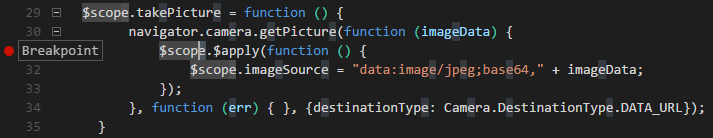
1. Defines a takePicture() function that can be called from the account tab.
2. Uses the Camera plugin to get a picture, by calling the navigator.camera.getPicture() API. This call to getPicture passes an option that says to return a DATA\_URL when a picture is taken. This URL is a Base-64 encoded string which can be passed directly into an <img> element without saving a file to the device. (the code that sets $scope.imageSource defines the string that is set in the img src attribute on the page).
3. That call to $scope.$apply() is needed with Angular to tell the system to update data bindings in the UI.

Finally, **run the application** again, **switch to the Account tab** and try your new button!

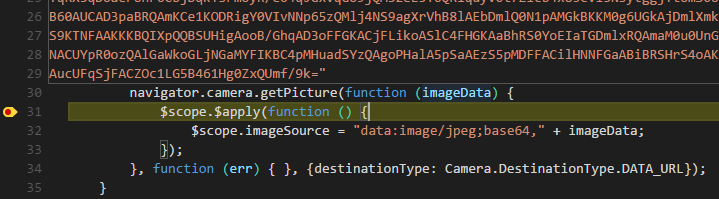
# Step 4: Debug the application

Visual Studio Code provides a JavaScript debugger that can be used to debug code running on Android and iOS devices & emulators. For a final step in this challenge, let’s use it.

With the controller.js file opened from the previous step, **set a debugger breakpoint by clicking in the margin next to the line that reads $scope.$apply**:



**Run the app, switch to the Account tab, and click on the Set My Picture button** and then **take a picture with the camera.** The debugger will stop at the breakpoint you just set. Now you have access to standard debugging features like the Call Stack, Locals & Watches, and the ability to inspect variables by hovering over them. For example, **hover over the imageData parameter** to see the Base-64 encoded image that will be shown in the UI.

****

# Summary

Congratulations on completing this Quick Start Challenge! You just:

* Created a simple mobile app using Apache Cordova and the Ionic framework.
* Tested the app on a local Android device or iOS simulator.
* Accessed a native device’s camera by using a Cordova plugin.
* Debugged JavaScript running in the app.

# Additional Resources

If you’re interested in learning more about Cordova and the Visual Studio Code Cordova extension, check out the following resources:

**Visual Studio Code:** <http://code.visualstudio.com>

**Visual Studio TACO Tools:** <http://www.taco.tools>

**Cordova Extension:** <http://aka.ms/VSCodeCordovaExtension>

**The Ionic Framework:** <http://www.ionicframework.com>

**Apache Cordova home page:** <http://www.cordova.io>