Creating a Cordova app using Ionic in Visual Studio 2015

# 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, Microsoft Visual Studio 2015 includes the Visual Studio Tools for Apache Cordova (also known as TACO). 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 and Visual Studio TACO, by building a simple Getting Started app.

**Note:** Due to hardware limitations, this code challenge focuses on developing against a Windows device. If you’d like to try building for an iOS or Android device, try this same challenge over in the Garage.

# Objectives

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

# Prerequisites

The following has already been setup for this code challenge:

* Windows 10
  + With Developer Mode enabled in (**Settings | Update & Security | For developers**)
* Visual Studio 2015 with Update 2
  + Include the **Apache Cordova features** by following the VS TACO Installation Guide at <http://aka.ms/VSTACOInstallGuide>
  + Include the **Universal Windows App Development Tools feature** of VS, as well.
* Ionic Project Templates for Visual Studio – <http://aka.ms/VSTACOIonicTemplates>

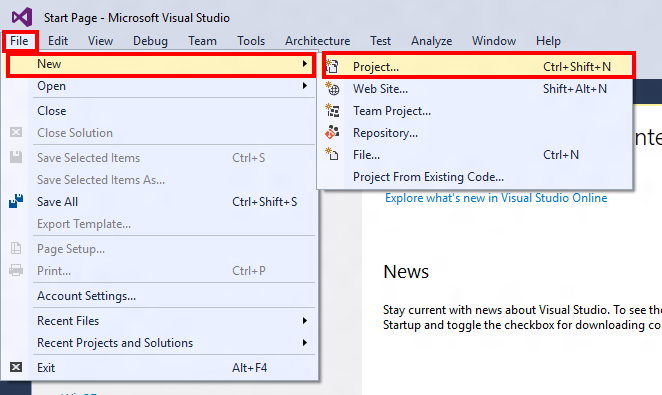
# This quick start is for you, if…

This Quick Start Challenge is for any developer already familiar with Visual Studio. Prior experience with web development isn't needed, but the experience and languages will make more sense to you if you are a web developer.

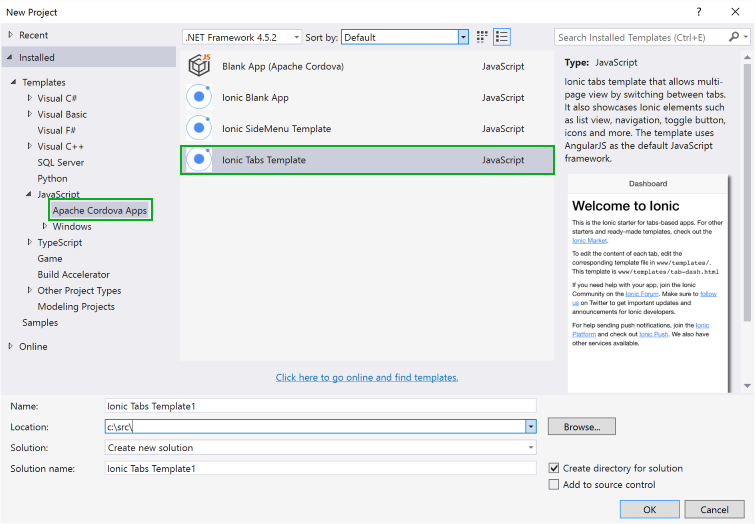
# Step 1: Create a new project in Visual Studio

Let’s start by creating a new project in Visual Studio.

First, launch **Visual Studio 2015**. Then, create a new project using the **File | New | Project...** menu.



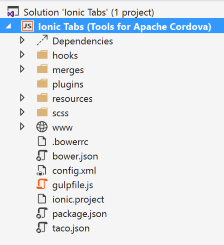
Browse to **Installed | Templates | JavaScript | Apache Cordova Apps** in the new project dialog and choose the **Ionic Tabs Template** as your project template.



Give your project a name and then click **OK**.

**Note**: This template 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>.

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

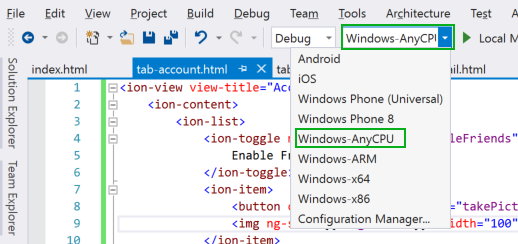


A few things worth calling out here:

* 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.

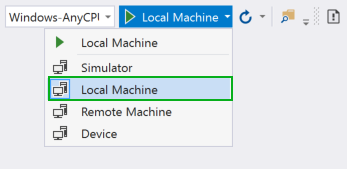
# Step 2: Run the app

Let's run the app on Windows 10 and see what it looks like. In the **Debug Toolbar**, choose the **Windows-AnyCPU platform from the platforms drop-down**.



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

Now, choose the **Local Machine** device target from the **Device Targets drop-down** located next to the **Platforms drop-down**.



Then click on the  button to start the app. A build step will immediately start downloading packages required for the app, and then taking the files in your project and building them into a Windows App (deployed as a .appx file).

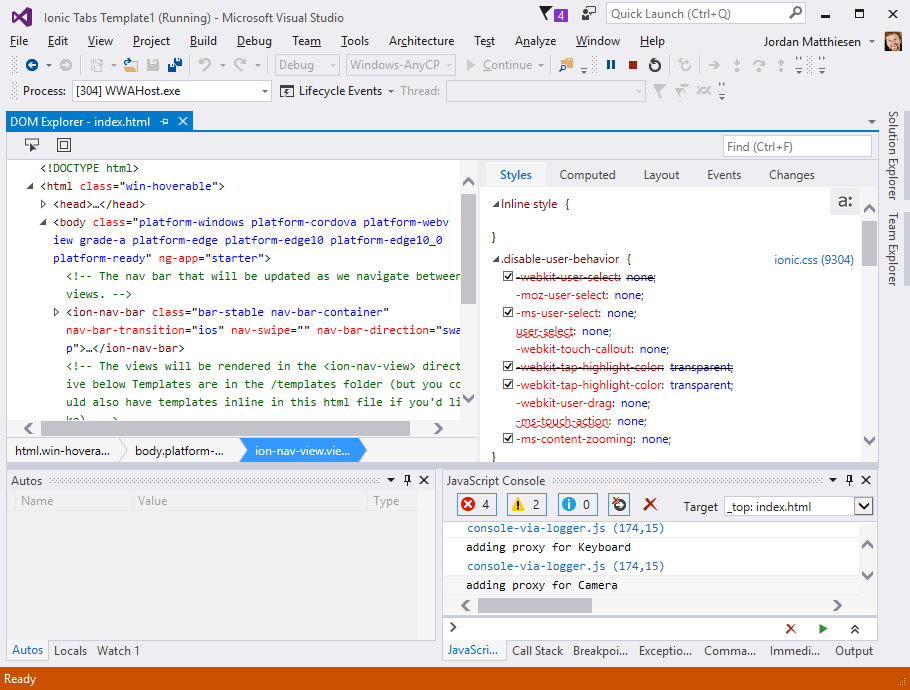
**Note:** This same Device Target drop-down is where you go to select emulators and devices for Windows, iOS, and Android device development.

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

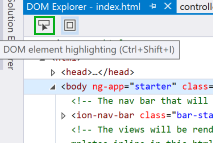
# 

# Step 3: Use the DOM Explorer

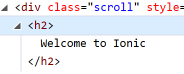
Back in Visual Studio, you'll see the **DOM Explorer and JavaScript Console** windows opened. These tools to help you debug your applications UI and functionality. The DOM Explorer lets you explore the layout of your app as its running, and experiment with markup and style changes. The JavaScript Console helps you diagnose errors in your JavaScript code.



To give you a sense of this tool, let’s make a quick change. In the DOM Explorer window, choose the **Select Element (****)** button.



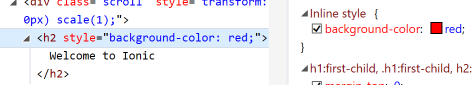
In your app, tap on the “Welcome to Ionic” text in your app. You'll see that the item you selected on the device is now selected in the DOM Explorer.



Now, in the **Styles** pain on the right-side of the screen, click in the inline style section to add the following code:

background-color: red;

When done, your DOM explorer window should look like the following and you'll see a red header in your app:



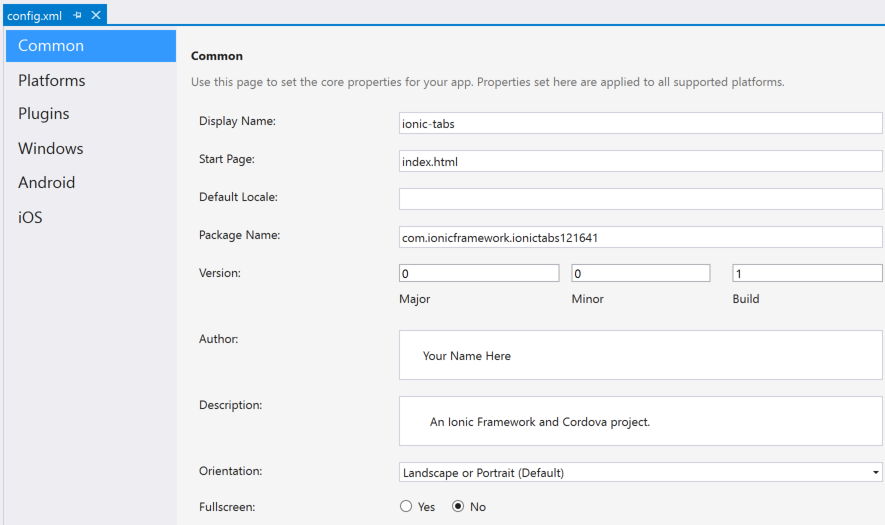
**Close the application** now, to reset the UI styles for the next section.

# Step 4: Access native device capabilities

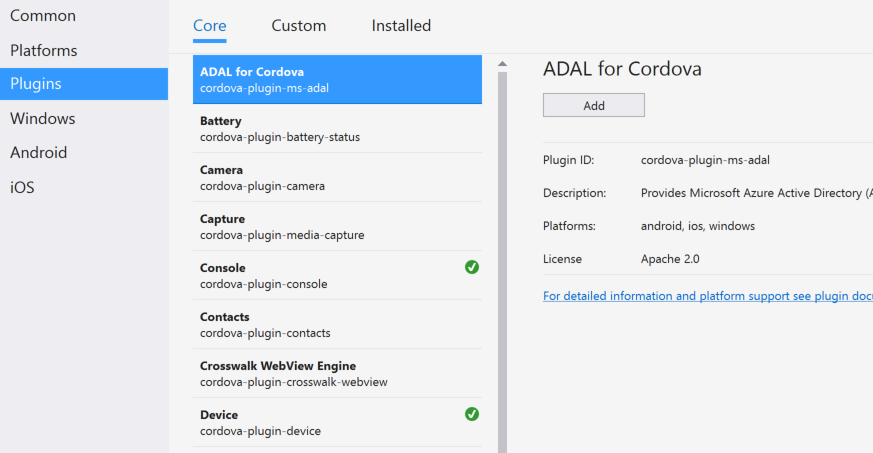
With Cordova, you can get at native device capabilities (such as the Camera) by using Plugins. Plugins provide a common JavaScript API that you use across device platforms, and the plugin author has done the work to implement the native functionality for each platform.

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.

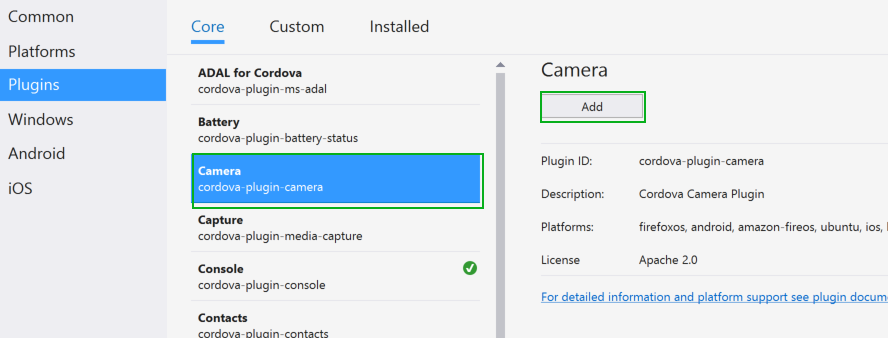
To get started, **open the config.xml file** in your project, you’ll see the configuration designer appear.



Click on the **Plugins** tab and you’ll see a list of recommended plugins.

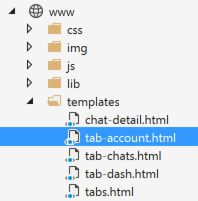


To add the Camera plugin to this project, click on the **Camera option** in the **list of Core plugins**, and then click the **Add** button.



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">

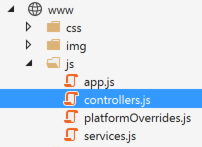
</ion-item>

</ion-list>

</ion-content>

</ion-view>

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});

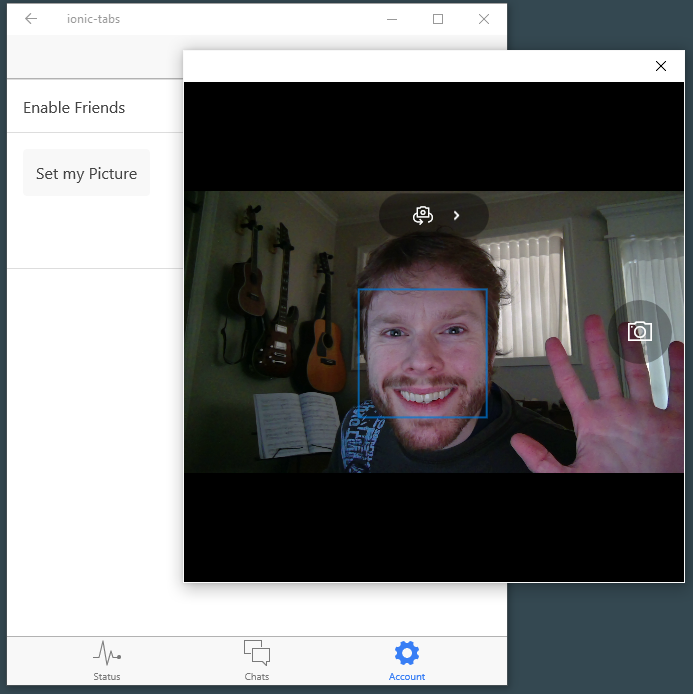
}

});

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.

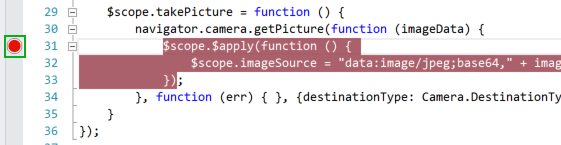
Finally, **run the application** again, **switch to the Account tab** and try your new button!



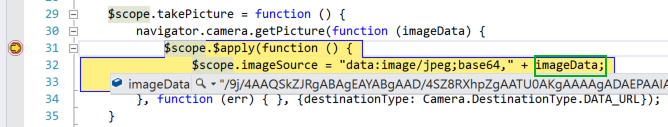
# Step 5: Debug the application

Along with the DOM Explorer and JS Console described earlier, Visual Studio provides a JavaScript debugger that can be used to debug code running on Android, iOS, and Windows 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 in Visual Studio 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 Windows device.
* Modified the HTML DOM, live, using the DOM Explorer.
* 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 Tools for Apache Cordova (TACO), check out the following resources:

**Visual Studio TACO documentation site:** <http://taco.visualstudio.com>

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