* **10 Steps To Publish Your First Android App On Google Play Store**

Google Play is the premier store for distributing an Android app. After sweating out hours on creating a perfect app, you are now ready to put it out in front of over 1 billion active Android users in more than 190 countries and territories around the world. And, to make the process of publishing your app as smooth as it can be on Google Play Store, here is a quick step by step guide. So, are you ready to publish your android app on Google Play Store?

1. **Create an account**

To publish your app in the Google Play Store, you need to have an account with Google. You might already have a personal email account with them, but it is better to have a separate one to manage your app(s). While registering your publisher account, you would have to pay a registration fee of 25 USD, using Google payments. After this, a verification mail would be sent to you; and then, you can sign in to your Developer Console, where all the action would take place.

1. **Familiarise yourself with Developer Console**

Google Play Developer Console is the starting point and the main dashboard for app publishing operations and tools. Before you go ahead get to know its interface, also familiarise yourself with the list of developer countries and merchant countries. The list of developer countries will tell you about all the locations where distribution to Google Play users is supported. And, if you want to sell apps, or have subscriptions or in-app purchases, then you need to review the list of merchant countries. Apart from this, take a look at the Developer Distribution Agreement, to avoid anything that violates the Google Play’s terms and conditions.

1. **Fill in the necessary account details**

After this, log in to complete your account details. For example, you need to provide your Developer Name, the name which would be displayed in the Google Play Store. After filling in the details, you will have to wait for anything between just a little and 48 (the official time frame) hours, for the Google Play Developer registration to be processed.

1. **Link your merchant account**

If you have a paid app, or one with subscriptions or in-app purchases, then you need to link your Google payments merchant account to your developer profile. Your linked account can be used for financial and tax identification as well as monthly payouts from sales.

1. **Upload your app**

When you are logged in to the Google Play Developer Console, click on “Add New Application” in the “All Applications” tab. Select the correct “Default Language” from the drop-down menu and then type the “Title” of the app, which is your app’s name with which it will appear in the Play Store. After this, select the “Upload APK” button to land on a new page, which would be the homepage for your app. Here, upload the pre-prepared “.apk” file of your app. The timeframe for uploading of the file will depend on its size. The app will remain in the drafts until and unless you publish it.

1. **Alpha and Beta testing the app**

Before you launch your app, it is essential to test it with a sample of end users to get their feedback; and the Google Play takes care of that as well. In the “APK” section of the Developer Console, you will find the options related to “Beta Testing” and “Alpha Testing”. After you have uploaded your app’s “.apk” file(s), you can use these options to receive a URL that can be shared with the testers. Using this link, testers can download your app’s alpha or beta version. Just remember, your testers cannot provide feedback and reviews on the app page; for this, you would have to get in touch with them yourself. You can use their feedback to optimise your app and make relevant changes in it before publishing it in the store.

1. **Provide details for store listing**

Now comes the most important part of uploading the app, as this is the place that will determine how it would be seen in Google Play. After uploading the “.apk” file of your app, go to the “Store Listing” tab. Over there, you need to add the details of the app, like a “Short description” (of 12 to 80 characters) and a “Full description” (of up to 100 characters) of the app. Along with this, add screenshots, a link of the promo video (if you have one), contact details (website name, email id, phone number, etc.), categorisation (application type category and content rating), and other such important details related to the app. After completing the relevant fields, press the “Save” button. You can update your store listing at any time, so do not panic if you have made some mistake or left out a field while filling up.

1. **Add pricing and distribution details**

Now, move to the next tab, which is “Pricing & Distribution” and select whether it is a “Free” or a “Paid” app. Also, select the distribution countries and check the boxes stating that your app complies with the content guidelines. If your app is an educational one, then you can put it in the limelight using “Google Play for Education” option or if it is compatible with an Android TV, then you can add a Leanback launch over here. Once you are done, save the changes and move on to the next step.

1. **Publishing the application**

When all the three tabs- “APK”, “Store Listing” and “Pricing & Distribution”- have been filled and there appears a green check mark next to them, you are all ready to publish the app in Google Play. All you need to do is, click the “Publish this app” button under the “Ready to Publish” drop-down menu at the top right corner of the Developer Console. After you have hit the button, a confirmation bar would show up stating that your app would appear shortly in the Google Play Store. Once your app is published, you can update it as often as you want. You can even make changes with the pricing, configuration and distribution options at any time.

1. **Device Filtering option**

These are a series of extra options that might not seem to be of much importance as you publish the app, but they can prevent your app from getting negative feedbacks. Through Google Play, you can control the distribution of your app according to the device features that are compatible with it. Allow only those devices to find your app in the Play Store that are compatible with your app. There is also an option to manually filter problematic or non-compatible devices, so make the most of it to stay on the top and filter out any negativities.

# Package Your Cordova App for Publishing to an App Store

When you build an application for deployment to an emulator, simulator, physical device or the Cordova Simulate browser, you're building a version of the application specifically crafted for local testing of the app. The app is usually built with debug information packaged into the executable, and the app is signed with a signing key which allows it to work on your local device, but not on any device.

Before you can build your application for deployment to any device through a public app store, you must first configure Visual Studio with the information it needs to complete the process of packaging and signing the application for deployment through an app store. In this article, you'll learn how to configure a project for deployment and create a deployment package for each target platform:

[Android](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#android)

[iOS](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios)

[Windows](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#windows)

## Android Applications

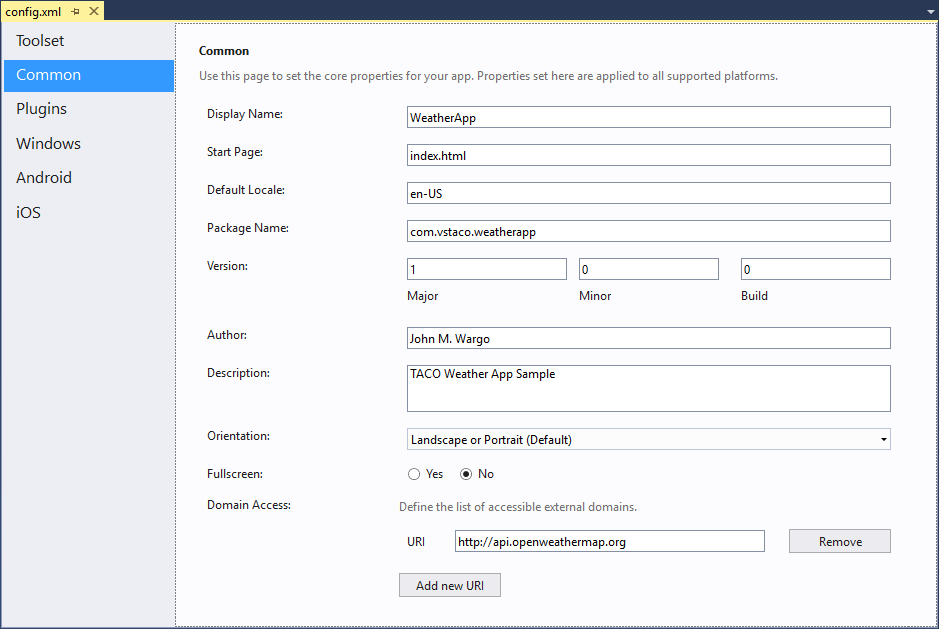
For Android applications, preparing your environment for publishing, and publishing the app requires that you complete the following steps:

1. [Modify Application Settings](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#android-1)
2. [Generate a Private Certificate](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#android-2)
3. [Modify the Android App Build Configuration](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#android-3)
4. [Create the Deployment Package](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#android-4)

### 1. Android: Modify Application Settings

As you prepare to publish your shiny new Cordova application, start in the application's configuration and make sure your settings for the application are correct. A Cordova app's settings are maintained in the project's config.xml file.

1. In the Visual Studio Solution Explorer, double-click the config.xml file to open the custom configuration editor shown in the following figure:

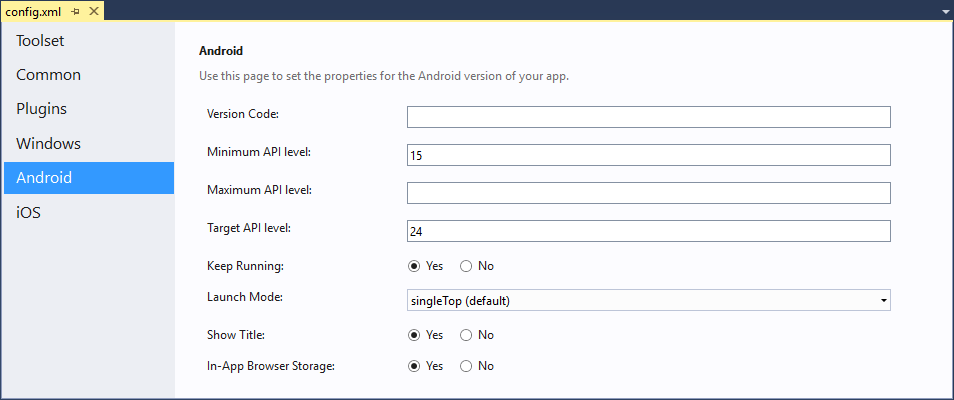


The editor's **Common** tab contains general settings for your app; populate the fields in the form with the appropriate values for your application. The critical settings for any Cordova application are:

* + **Display Name**: the application's public name; this is the how the app will appear in the app store and on the target mobile device's home screen and application listing. Populate this field with a brief word or phrase that describes your app, keeping in mind that app tiles on a device's home screen don't leave much room for text. The value you enter here is added to the config.xml in the <name> element.
  + **Package Name**: the unique identifier for this application. Developers typically populate this field with a combination of the developer's company domain in [**reverse domain name notation**](https://en.wikipedia.org/wiki/Reverse_domain_name_notation) plus the short name for the application. The value you provide here is added to the widget element's id attribute as shown in the following example"
  + **Domain Access**: Manages a list of domains that the application can access; the values you enter here are added as access elements to the config.xml as shown in the following example:

The purpose of most other settings clear from the title, but you can find more information about them here: [The config.xml File](http://cordova.apache.org/docs/en/latest/config_ref/index.html).

1. Switch to the editor's **Android** tab to set Android-specific settings for the application. These settings control the conditions under which the application runs on an android device.



Each input field on the form corresponds to a specific entry in the project's config.xml file:

* + **Version Code**: android-versionCode - a string value, used to set the project's [version code](https://developer.android.com/studio/publish/versioning.html) for the app. See [setting the version code](http://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html#setting-the-version-code) for additional information.
  + **Minimum API level**: android-minSdkVersion - an integer value that represents the minimum device API level for the application. The application will not install on a device if its OS API level is lower than this value.
  + **Maximum API level**: android-maxSdkVersion - an integer value that represents the maximum device API level for the application. The application will not install on a device if its OS API level is higher than this value.
  + **Target API level**: android-targetSdkVersion - an integer value representing the application's [API compatibility](https://developer.android.com/guide/topics/manifest/uses-sdk-element.html).
  + **Keep Running**: KeepRunning - a Boolean value used to determine whether the application stays running in the background after the pause event fires. A false value does not kill the app after a pause event, but simply halts execution of code within the Cordova WebView while the app is in the background.
  + **Lanch Mode**: AndroidLaunchMode - a string value that sets the Activity android:launchMode attribute in the application. This changes what happens when the app is launched from app icon or intent and is already running. Valid values are **standard**, **singleTop**, **singleTask**, and **singleInstance**.
  + **Show Title**: ShowTitle - a Boolean value that controls whether the application displays the app title at the top of the application's main screen.
  + **In-App Browser Storage**: InAppBrowserStorageEnabled - Controls whether pages opened within an InAppBrowser window can access the same localStorage and WebSQL storage as pages opened with the default browser.

You can read about each configuration option in the Cordova [config.xmlreference](http://cordova.apache.org/docs/en/latest/config_ref/index.html#preference) guide.

### 2. Android: Generate a Private Certificate

When running Android applications using the Android SDK (which Visual Studio uses under the covers), applications are signed with a **debug certificate generated by the Android SDK tools**. Before you can sign Android applications for deployment via other means, you must use a signing certificate for your organization. If you already have a certificate you would like to use to sign your Android applications, feel free to skip the remainder of this section.

Note

Certificates are stored in a **keystore**, if you already have a keystore on your system you'd like to use to store your certificate, you'll need the keystore location and credentials before you continue.

**To create a signing certificate, complete the following steps:**

Open a Windows Command Prompt.

If your existing keystore is in a protected folder (like c:\ for example), or you'll be generating a keystore in a protected folder, you'll need to open the command prompt in Administrator mode for these steps to complete successfully.

If your system is configured with the Java SDK bin folder on the system PATH, then skip to the next step. You can confirm this by typing javac in the command window and pressing enter. If you receive an error message, the JDK is **not** on the path. If you see the Java Compiler help page, then you're in good shape and can skip this step.

In the Command Prompt, change directories to the Java SDK's bin folder. If your development system has the %JAVA\_HOME% environment variable set, then it should be %JAVA\_HOME%\bin. You can also switch to the SDK folder using the complete path (such as: C:\Program Files\Java\jdk1.8.0\_111\bin).

1. In the Command Prompt, execute the following command:

Copy

keytool -genkeypair -v -keystore FILE-PATH\MY-KEYSTORE-NAME.keystore -alias MY-ALIAS -keyalg RSA -keysize 2048 -validity 10000

Replacing FILE-PATH\MY-KEYSTORE-NAME.keystore and MY-ALIAS with the appropriate values for your needs. If you have an existing keystore you want to use, substitute your keystore file path for DRIVE:\FILE-PATH\MY-KEYSTORE-NAME.keystore in the example

For example, if you wanted to write the keystore to the system's dev folder in a keystore file called jd-release-key.keystore and using an alias of android-key for the generated key, you would issue the following command:

Copy

keytool -genkeypair -v -keystore c:\dev\jd-release-key.keystore -alias android-key -keyalg RSA -keysize 2048 -validity 10000

The Java keytool application will launch and prompt you for a series of values it needs to create the keystore and generate the keypair:

Copy

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]: Joe Developer

What is the name of your organizational unit?

[Unknown]: NA

What is the name of your organization?

[Unknown]: Company

What is the name of your City or Locality?

[Unknown]: Charlotte

What is the name of your State or Province?

[Unknown]: NC

What is the two-letter country code for this unit?

[Unknown]: US

Is CN=Joe Developer, OU=NA, O=Company, L=Charlotte, ST=NC, C=US correct?

[no]: y

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days

for: CN=Joe Developer, OU=NA, O=Company, L=Charlotte, ST=NC, C=US

Enter key password for <android-key>

(RETURN if same as keystore password):

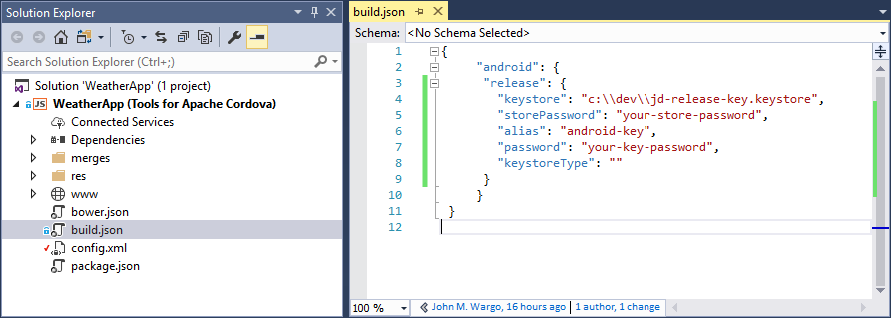
[Storing c:\dev\jd-release-key.keystore]

If you want more detail about this process, see the Android developer documentation here: [Signing your applications](http://developer.android.com/tools/publishing/app-signing.html).

### 3. Android: Modify the Android App Build Configuration

Now that you have a keystore and a signing certificate, you must configure your project to use them.

1. In **Solution Explorer**, expand the project folder, and double-click on the project's build.json file. The build.json file opens in the code editor:



##### **Note**

If the build.json file is missing from your project, its likely that your project was created with an earlier version of Apache Cordova; you should create that file manually (and populate it with content shown in step 2).

1. Populate the build.json file with the keystore and key details:

JavaScriptCopy

{

"android": {

"release": {

"keystore":"c:\\dev\\jd-release-key.keystore",

"storePassword":"your-store-password",

"alias":"android-key",

"password":"your-key-password",

"keystoreType":""

}

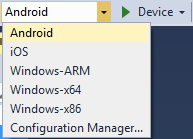
}

}

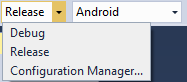
### 4. Android: Create the Deployment Package

The final step involves creating a **Release** build of the Cordova Application.

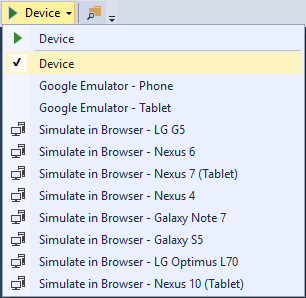
1. On the Standard toolbar, choose the **Android** platform.



1. Choose the **Release** build configuration.



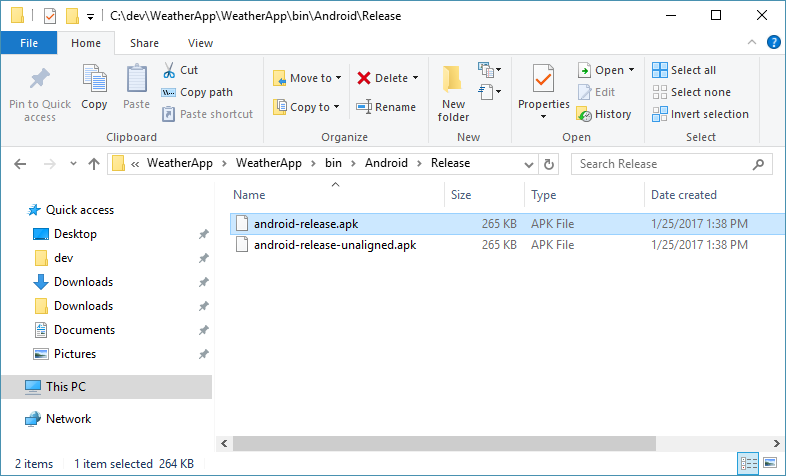
1. Choose one of the Android emulators or a physical device.



##### **Note**

**Do not select** one of the **Simulate in Browser** options, they don't generate a native application binary. Choose only an **Android emulator**or **Device**.

1. In the **Build** menu, select **Build Solution**. This creates a release build of the application, a file with an .apk file extension. This is the file that you'll upload to the store when you deploy the application.
2. When the build completes, look for the .apk file; you'll find it in the project's bin/Android/Release/ folder. When uploading the app to the app store, be sure to select the file that **does not** include the word unaligned in the file name.



At this point, you have a release version of the application all ready to distributed through the Google Play App Store.

## iOS Applications

For iOS applications, preparing your environment for publishing, and publishing the app requires that you complete the following steps:

1. [Request a Distribution Certificate](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios-1)
2. [Modify Application Settings](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios-2)
3. [Create the Deployment Package](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios-3)
4. [Submit Your App to the App Store](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios-4)

### 1. iOS: Request a Distribution Certificate

Apple uses **Distribution Certificates** to identify a developer, development team, or organization. You'll need a distribution certificate to deploy applications through the Apple App Store. If your team already has one and you want to use it, refer to [How to share an iOS distribution certificate](http://www.ironpaper.com/webintel/articles/how-to-share-an-ios-distribution-certificate/). Then, skip straight to the [Modify Application Settings](https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/publishing/publish-to-a-store#ios-2) section of this document.

##### **Note**

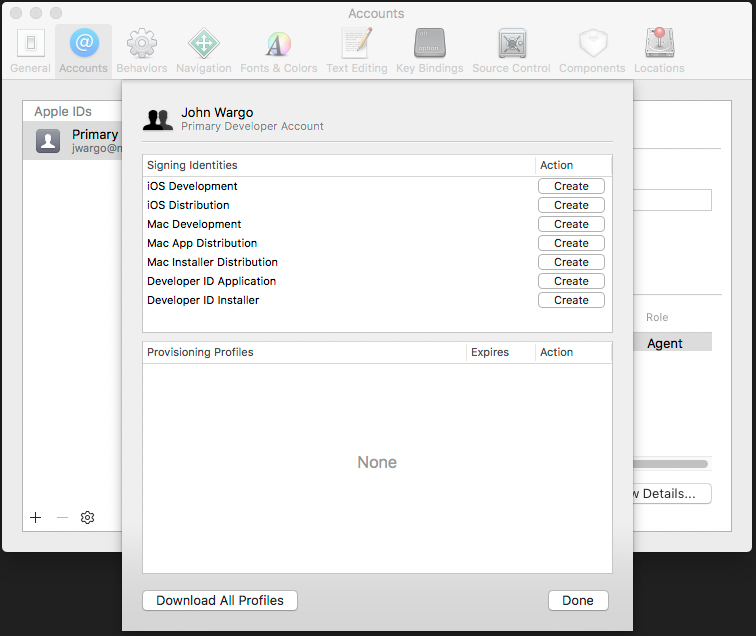
Building iOS applications, requesting distribution certificates, and deploying applications to the app store all require the use of Apple's Xcode development environment running on a computer system running [macOS](http://www.apple.com/macos/). You'll also need an active [Apple iOS developer program account](https://developer.apple.com/ios/). Be sure you have the appropriate iOS development environment setup before continuing.

**To create a distribution certificate, complete the following steps:**

1. Switch to your Macintosh system and open the Xcode IDE.
2. In the system's menu, select **Xcode** -> **Preferences**.

If you haven't done so already, add your developer account Apple ID to the system's configuration. See [Adding an Apple ID to Your Accounts](https://developer.apple.com/library/ios/recipes/xcode_help-accounts_preferences/articles/add_appleid.html) for instructions.

1. In the **Accounts** section, select your developer account, then click the **View Details** button.
2. In the account details window, under **Signing identities**, look for the item labeled **iOS Distribution**. If there is a **Create** button to the right the iOS Distribution entry, click the button to create and download the signing identity. If the Create button isn't shown, that means this step has already been completed.



Looking for more information about signing identities? See [Creating Signing Identities](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html#//apple_ref/doc/uid/TP40012582-CH31-SW6) (Optional reading).

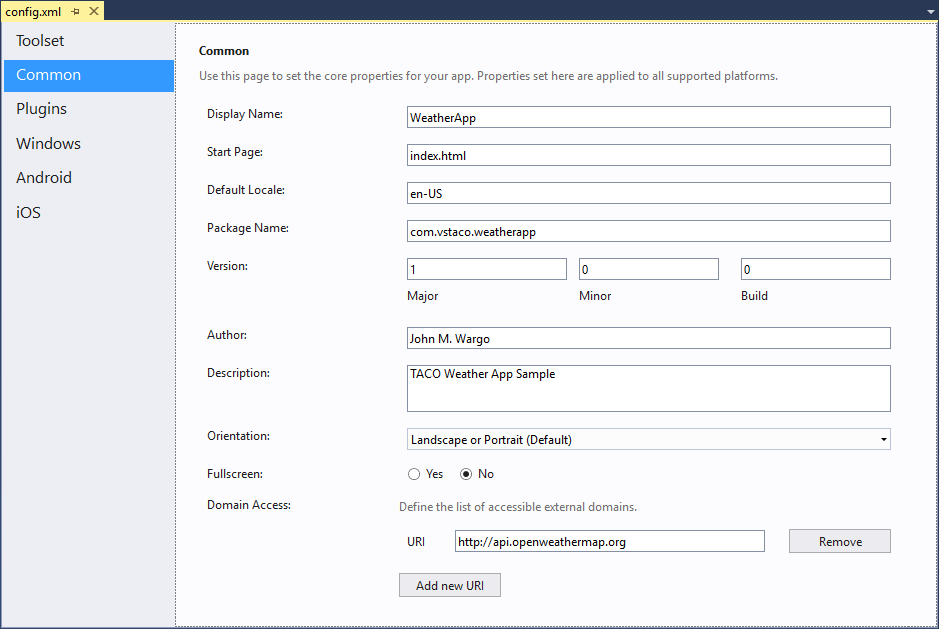
Xcode will submit a request to generate the distribution profile and hide the button you clicked to indicate that the distribution profile exists.

1. Click the **Done** button to close the account details window.

### 2. iOS: Modify Application Settings

As you prepare to publish your shiny new Cordova application, start in the application's configuration and make sure your settings for the application are correct. A Cordova app's settings are maintained in the project's config.xml file.

1. In the Visual Studio Solution Explorer, double-click the config.xml file to open the custom configuration editor shown in the following figure:

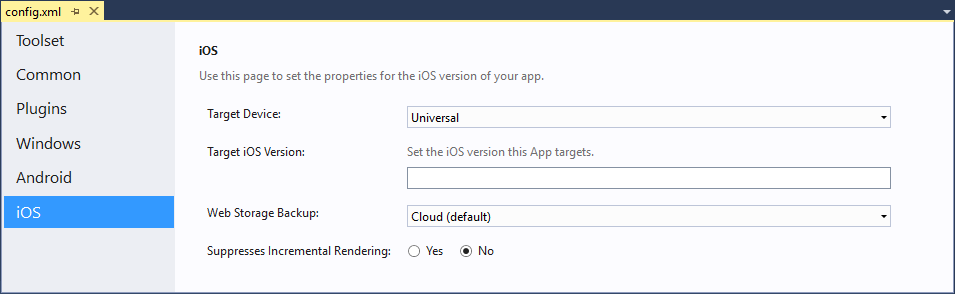


The editor's **Common** tab contains general settings for your app; populate the fields in the form with the appropriate values for your application. The critical settings for any Cordova application are:

* + **Display Name**: the application's public name; this is the how the app will appear in the app store and on the target mobile device's home screen and application listing. Populate this field with a brief word or phrase that describes your app, keeping in mind that app tiles on a device's home screen don't leave much room for text. The value you enter here is added to the config.xml in the <name> element.
  + **Package Name**: the unique identifier for this application. Developers typically populate this field with a combination of the developer's company domain in [**reverse domain name notation**](https://en.wikipedia.org/wiki/Reverse_domain_name_notation) plus the short name for the application. The value you provide here is added to the widget element's id attribute as shown in the following example"
  + **Domain Access**: Manages a list of domains that the application can access; the values you enter here are added as access elements to the config.xml as shown in the following example:

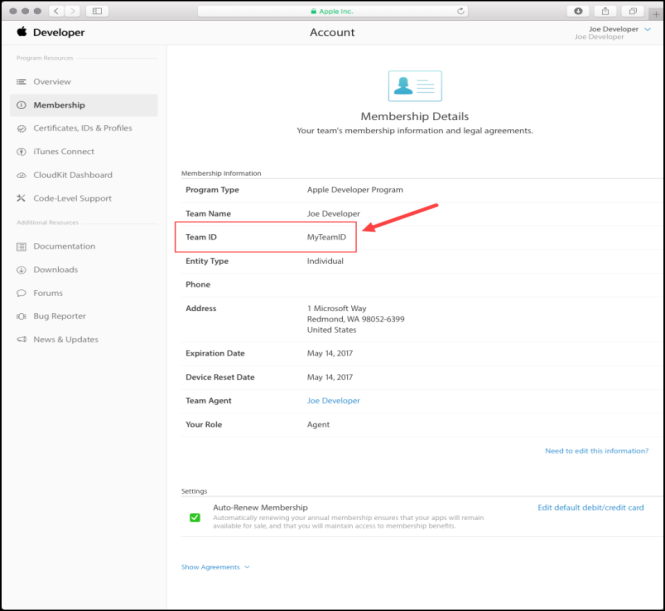
The purpose of most other settings clear from the title, but you can find more information about them here: [The config.xml File](http://cordova.apache.org/docs/en/latest/config_ref/index.html).

1. Switch to the **iOS** tab to set application settings specific to the iOS platform. Populate the fields on the form using the appropriate values for your application:
   * **Target Device**: Used to specify the type of device the application targets; valid options are: **handset**, **tablet**, and **universal**.
   * **Target iOS Version**: Sets the MinimumOSVersion in the generated .ipa file. For more information on this setting, refer to [Configuring a Project for SDK-Based Development](<https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/cross_development/Configuring/configuring.html>).
   * **Web Storage Backup**: Used to specify how iOS backs up application data stored using web storage. Available options are: **none**, **local**, or **cloud**. The cloud option backs up application data using iCloud. The local option enables local backups via iTunes sync. The none option disables storage backup.
   * **Suppress Incremental Rendering**: When set to true, screen rendering is delayed until all content has been received.



1. Even though Xcode manages provisioning and signing, you won't be opening Xcode to configure it for your project, this is all handled behind the scenes by Visual Studio TACO. Instead, you'll configure your **iOS Developer credentials**using the Cordova project's build.json file. During the build process, the Cordova CLI (invoked by Visual Studio TACO) copies settings for the iOS build from the build.json file to the Xcode project's configuration. Next, the remotebuild process (described in detail in the TACO [iOS Setup Guide](https://taco.visualstudio.com/en-us/docs/vs-taco-2017-ios-guide/)) executes the Xcode command-line tools to build and sign the app using the settings you provided.

To configure the iOS build process, you only need one piece of information, the Team ID for your Apple Developer program account. Open your browser of choice and navigate to <https://developer.apple.com/account>. After you've authenticated to the site, open the **Membership** page shown in the following figure. Note the Team ID from the page (it will be a series of letters and numbers, **not** MyTeamID as shown in the figure).



Next, open the Cordova project's build.json file, and add the following JSON object to the file, replacing MyTeamID with the correct value for your account:

JSONCopy

"ios": {

"debug": {

"developmentTeam": "MyTeamID"

},

"release": {

"developmentTeam": "MyTeamID",

"codeSignIdentity": "iPhone Developer",

"packageType": "ad-hoc"

}

}

Leave codeSignIdentity and packageType alone as those values are already set as needed. The data you'll be adding must coencide with the Android project settings that are probably already in the file. The resulting JSON object should look something like the following:

JSONCopy

{

"android": {

"release": {

"keystore":"c:\\dev\\jd-release-key.keystore",

"storePassword":"your-store-password",

"alias":"android-key",

"password":"your-key-password",

"keystoreType":""

}

},

"ios": {

"debug": {

"developmentTeam": "MyTeamID"

},

"release": {

"developmentTeam": "MyTeamID",

"codeSignIdentity": "iPhone Developer",

"packageType": "ad-hoc"

}

}

}

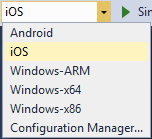
##### **Note**

Don't forget to include the comma between the android and iosobjects.

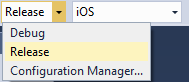
### 3. iOS: Create the Deployment Package

At this point, you're all ready to build the app to generate the package that you will submit to the app store for distribution.

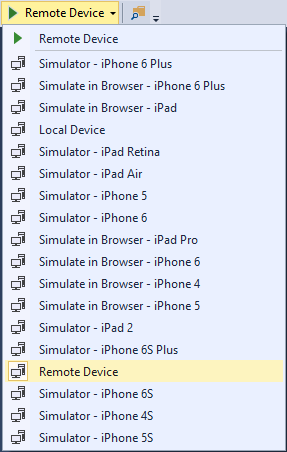
1. On your Macintosh system, make sure that the remote agent is running. Refer to the TACO [iOS Setup Guide](https://taco.visualstudio.com/en-us/docs/vs-taco-2017-ios-guide/) for details.
2. With the Cordova project open in Visual Studio, on the Standard toolbar, choose the **iOS** platform.



1. Choose the **Release** build configuration.

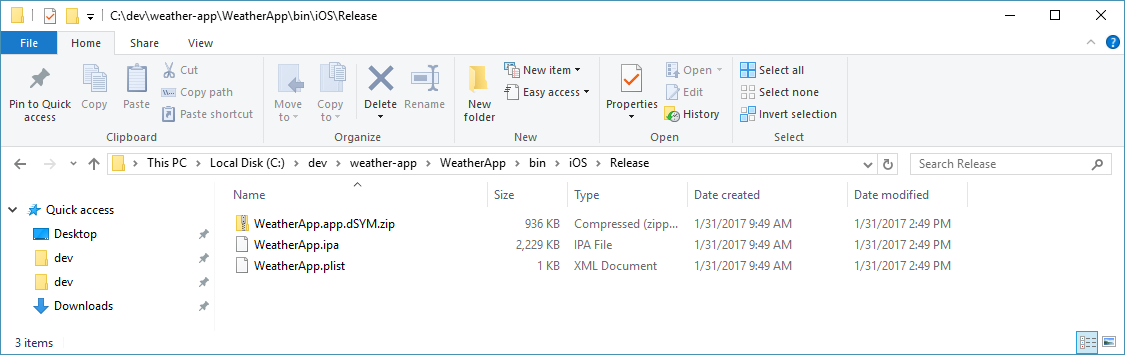


1. For execution target, select **Remote Device**.



1. On the **Build** menu, choose **Build Solution**.

This starts a build on the remotebuild agent and uses the distribution certificate and a newly generated provisioning profile to build a release signed iOS Application Archive (.ipa) file. You can find the resulting file in the bin/iOS/Release folder of your project.



### 4. iOS: Submit Your App to the App Store

1. Copy the application files onto a folder on your Mac.
2. Follow Apple's procedures for submitting the app to the App Store.