

Part 3 - Publishing an Application on Google Play

Although there are many app markets for distributing an application, Google Play is arguably the largest and most visited store in the world for Android apps. Google Play provides a single platform for distributing, advertising, selling, and analyzing the sales of an Android application.

This section will cover topics that are specific to Google Play, such as registering to become a publisher, gathering assets to help Google Play promote and advertise your application, guidelines for rating your application on Google Play, and using filters to restrict the deployment of an application to certain devices.

Requirements

To distribute an application through Google Play, a developer account must be created. This only needs to be performed once, and does involve a one time fee of \$25 USD.

All applications need to be signed with a cryptographic key that expires after October 22, 2033.

The maximum size for an APK published on Google Play is 100MB. If an application exceeds that size, Google Play will allow extra assets to be delivered through *APK Expansion Files*. Android Expansion files permit the APK to have 2 additional files, each of them up to 2GB in size. Google Play will host and distribute these files at no cost. Expansion files will be discussed in another section.

Google Play is not globally available. Some locations may not be supported for the distribution of applications.

Becoming a Publisher

In order to publish applications on Google play, it is necessary to have a publisher account. To sign up for a publisher account follow these steps:

1. Visit the [Google Play Developer Console](#).
2. Enter basic information about your developer identity.
3. Read and accept the Developer Distribution Agreement for your locale.
4. Pay the \$25 USD registration fee.
5. Confirm verification by e-mail.
6. After the account has been created, it is possible to publish applications using Google Play.

Google Play does not support all countries in the world. The most up to date lists of countries can be found in the following links:

1. [Supported Locations for Developer & Merchant Registration](#) – This is a list of all countries where developers may register as merchants and sell paid applications.
2. [Supported Locations for distribution to Google Play users](#) – This is a list of all countries where applications may be distributed.

Preparing Promotional Assets

In order to effectively promote and advertise an application on Google Play, Google allows developers to submit promotional assets such as screenshots, graphics, and video to be submitted. Google Play will then use those assets to advertise and promote the application.

Launcher Icons

A *launcher icon* is a graphic that represents an application. Each launcher icon should be a 32-bit PNG with an alpha channel for transparency. An application should have icons for all of the generalized screen densities as outlined in the list below:

- **ldpi** (120dpi) – 36 x 36 px
- **mdpi** (160dpi) – 48 x 48 px
- **hdpi** (240dpi) – 72 x 72 px
- **xhdpi** (320dpi) – 96 x 96 px

Launcher icons are the first things that a user will see of applications on Google Play, so care should be taken to make the launcher icons visually appealing and meaningful.

Tips for Launcher Icons:

1. **Simple and uncluttered**– Launcher icons should be kept simple and uncluttered. This means excluding the name of the application from the icon. Simpler icons will be more memorable, and will be easier to distinguish at the smaller sizes.
2. **Icons should not be thin**– Overly thin icons will not stand out well on all backgrounds.
3. **Use the alpha channel**– Icons should make use of the alpha channel, and should not be full-framed images.

High Resolution Application Icons

Applications on Google Play require a high fidelity version of the application icon. It is only used by Google Play, and does not replace the application launcher icon. The specifications for the high-resolution icon are:

1. 32-bit PNG with an alpha channel
2. 512 x 512 pixels
3. Maximum size of 1024KB

The [Android Asset Studio](#) is a helpful tool for creating suitable launcher icons and the high-resolution application icon.

Screen Shots

Google play requires a minimum of two and a maximum of eight screen shots for an application. They will be displayed on an application's details page in Google Play.

The specs for screen shots are:

1. 24 bit PNG or JPG with no alpha channel
2. 320w x 480h or 480w x 800h or 480w x 854h. Landscaped images will be cropped.

Promotional Graphic

This is an optional image used by Google Play:

1. It is a 180w x 120h 24 bit PNG or JPG with no alpha channel.
2. No border in art.

Feature Graphic

Used by the featured section of Google Play. This graphic may be displayed alone without an application icon.

1. 1024w x 500h PNG or JPG with no alpha channel and no transparency.
2. All of the important content should be within a frame of 924x500. Pixels outside of this frame may be cropped for stylistic purposes.
3. This graphic may be scaled down: use large text and keep graphics simple.

Video Link

This is a URL to a YouTube video showcasing the application. The video should be 30 seconds to 2 minutes in length and showcase the best parts of your application.

Publishing to Google Play

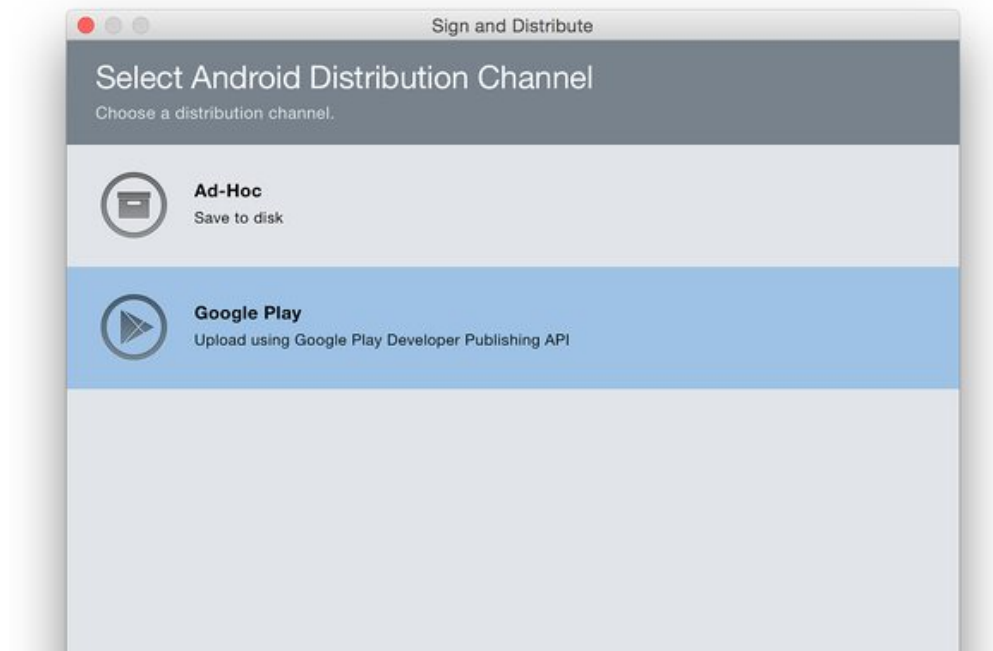
[[ide name="xs"]]

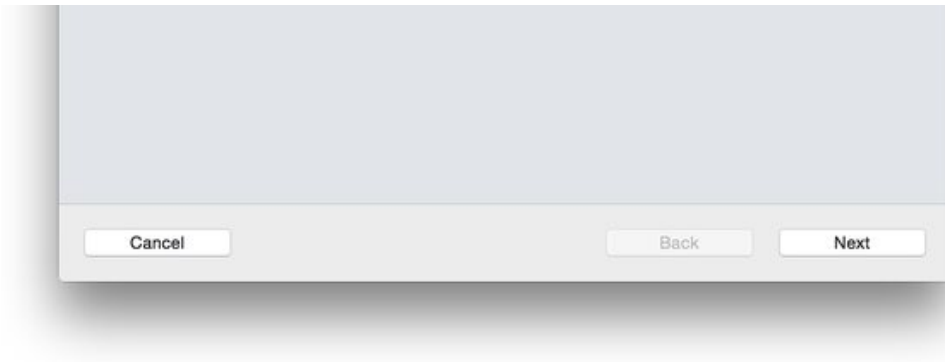
Xamarin Studio 5.9 introduced an integrated workflow for publishing apps to Google Play. If you are using a version of Xamarin Studio earlier than 5.9, you must manually upload your APK via the Google Play Developer Console and then use the **Publish to Google Play** dialog for subsequent APK updates. Also, you must have at least one APK already uploaded before you can use **Publish to Google Play**. If you have not yet uploaded your first APK, you must upload it manually. For information about how to manually upload an APK, see [Manually Uploading the APK](#).

[Part 2](#), discussed creatin a new certificate for signing Android apps. The following steps outline how to publish a Xamarin.Android app to Google Play:

1. Sign into your Google Play Developer account to create a new project that is linked to your Google Play Developer account.
2. Create an *OAuth Client* that authenticates your app.
3. Enter the resulting *Client ID* and *Client secret* into Xamarin Studio.
4. Register your account with Xamarin Studio.
5. Sign the application with your certificate.
6. Publish your signed application to Google Play.

In [Archive for Publishing](#), the **Sign and Distribute...** dialog presented two choices for distribution. Select **Google Play** and click **Next**:





In the **Google Play API Account** dialog, you must provide the *Client ID* and *Client secret* that provides API access to your Google Play Developer account:

Sign and Distribute

Google Play API Account

Authorize access to your Google Play Android Developer Account.


Account Description

Client Id

Client Secret

✖ You need to provide API access to your Google Play Developer account.

1. [Log in to your Google Play Developer account](#) and create a new project in the 'API access' tab in Settings. This project will be linked to your Google Play Developer account.
2. Create an **OAuth Client**.
3. Enter the **Client ID** and **Client Secret**.
4. **Register** your account.

 [More information](#)

The next section explains how to create a new Google API project and generate the needed *Client ID* and *Client secret*.

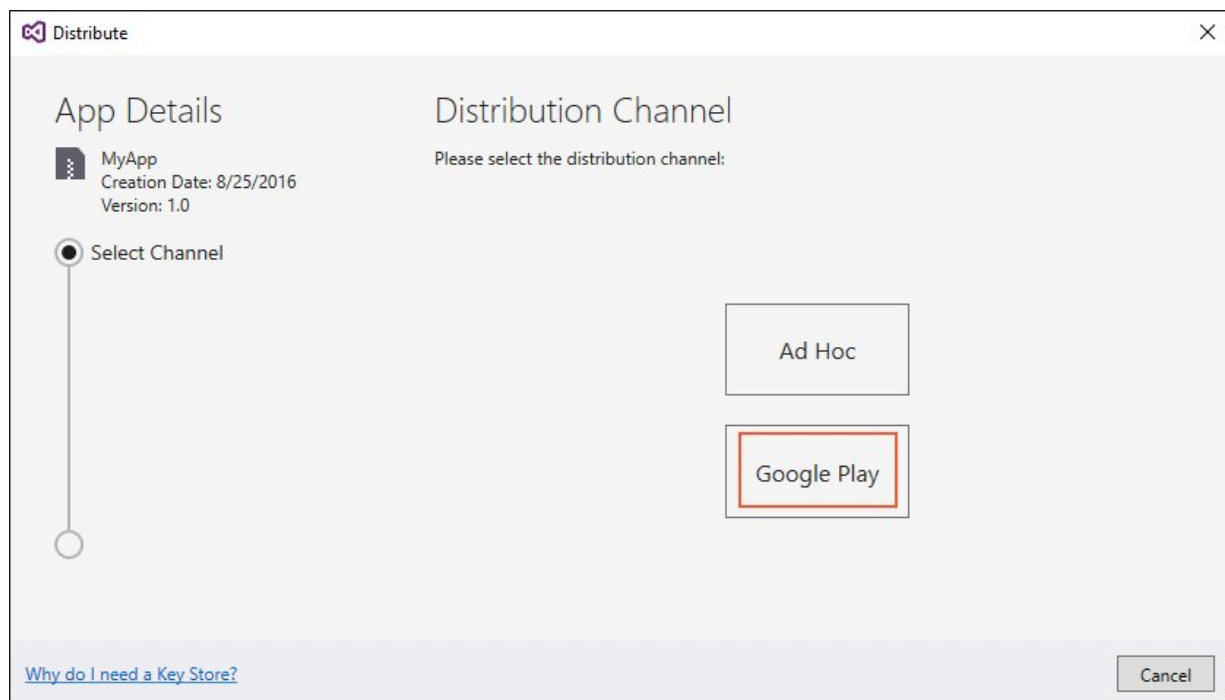
[[/ide]]

[[ide name="vs"]]

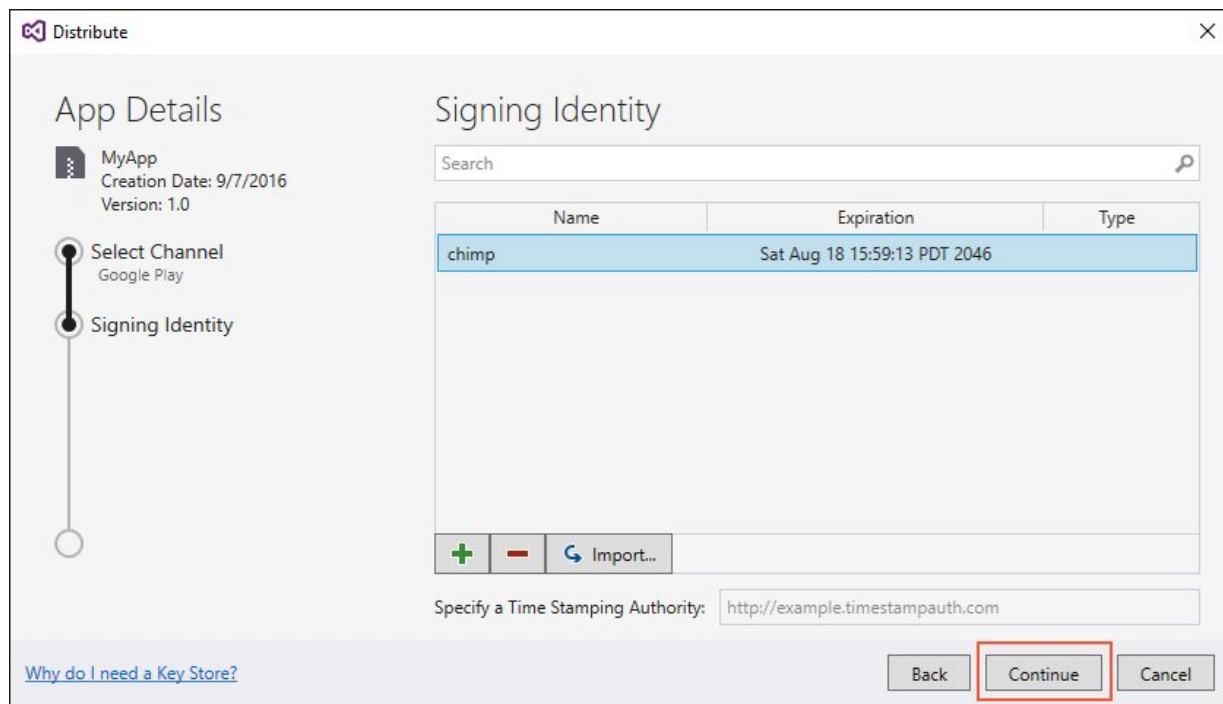
Xamarin Android 7.0 introduces an integrated workflow for publishing apps to Google Play from Visual Studio. If you are using a version of Xamarin Android earlier than 7.0, you must manually upload your APK via the Google Play Developer Console. Also, you must have at least one APK already uploaded before you can use the integrated workflow. If you have not yet uploaded your first APK, you must upload it manually. For more information, see [Manually Uploading the APK. Part 2](#), explained how to create a new certificate for signing Android apps. The next step is to publish a signed app to Google Play:

1. Sign into your Google Play Developer account to create a new project that is linked to your Google Play Developer account.
2. Create an **OAuth Client** that authenticates your app.
3. Enter the resulting Client ID and Client secret into Visual Studio.
4. Register your account with Visual Studio.
5. Sign the app with your certificate.
6. Publish your signed app to Google Play.

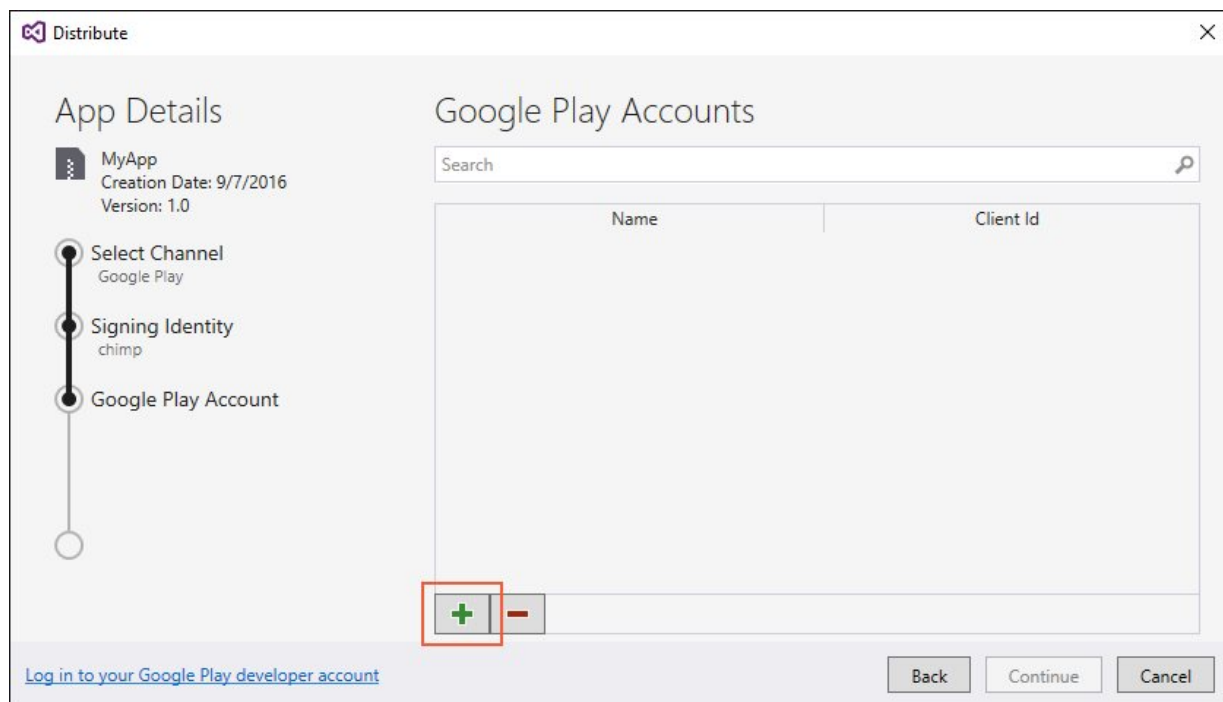
In [Archive for Publishing](#), the **Distribution Channel** dialog presented two choices for distribution: **Ad Hoc** and **Google Play**. If the **Signing Identity** dialog is displayed instead, click **Back** to return to the **Distribution Channel** dialog. Select **Google Play** and click **Next**:



In the **Signing Identity** dialog, select the identity created in [Part 2](#) and click **Continue**:

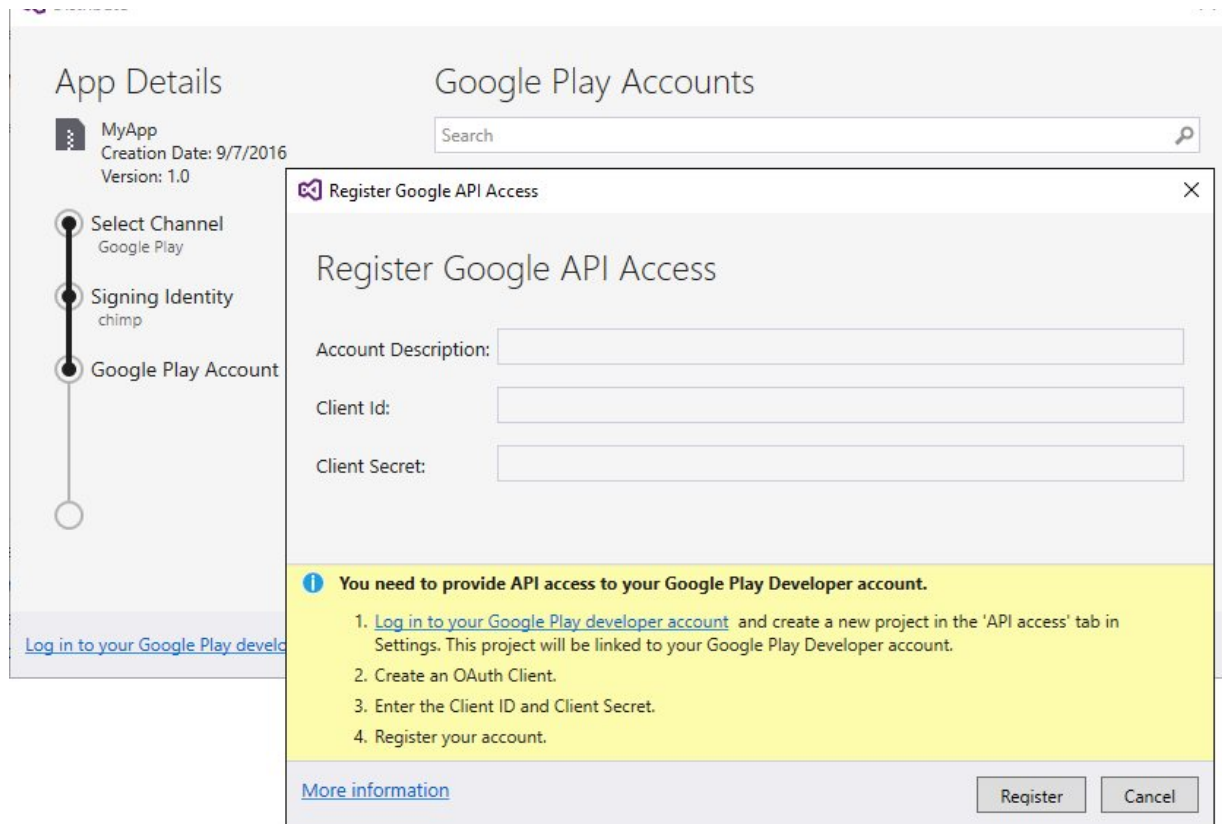


In the **Google Play Accounts** dialog, click the + button to add a new Google Play Account:



In the **Register Google API Access** dialog, you must provide the *Client ID* and *Client secret* that provides API access to your Google Play Developer account:



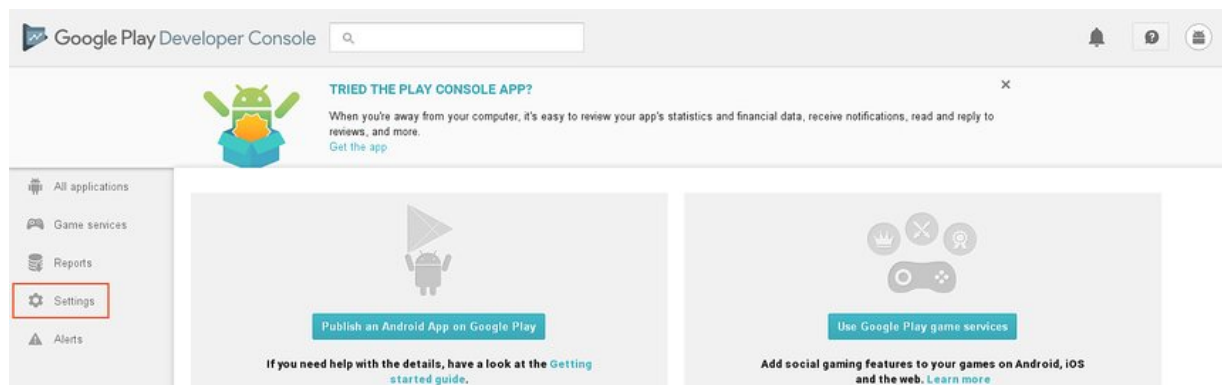


The next section explains how to create a new Google API project and generate the needed *Client ID* and *Client secret*.

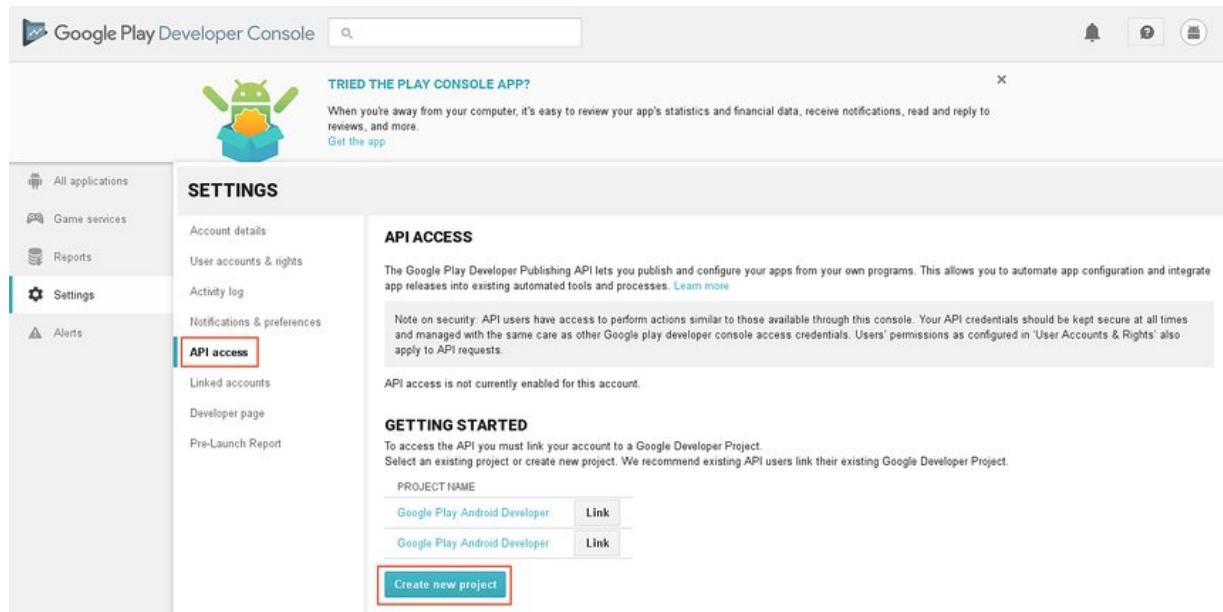
[[/ide]]

Create a Google API Project

First, sign into your [Google Play Developer account](#). If you do not already have a Google Play Developer account, see [Get Started with Publishing](#). Also, the Google Play Developer API [Getting Started](#) explains how to use the Google Play Developer API. After you sign into the Google Play Developer Console, click **Settings**:

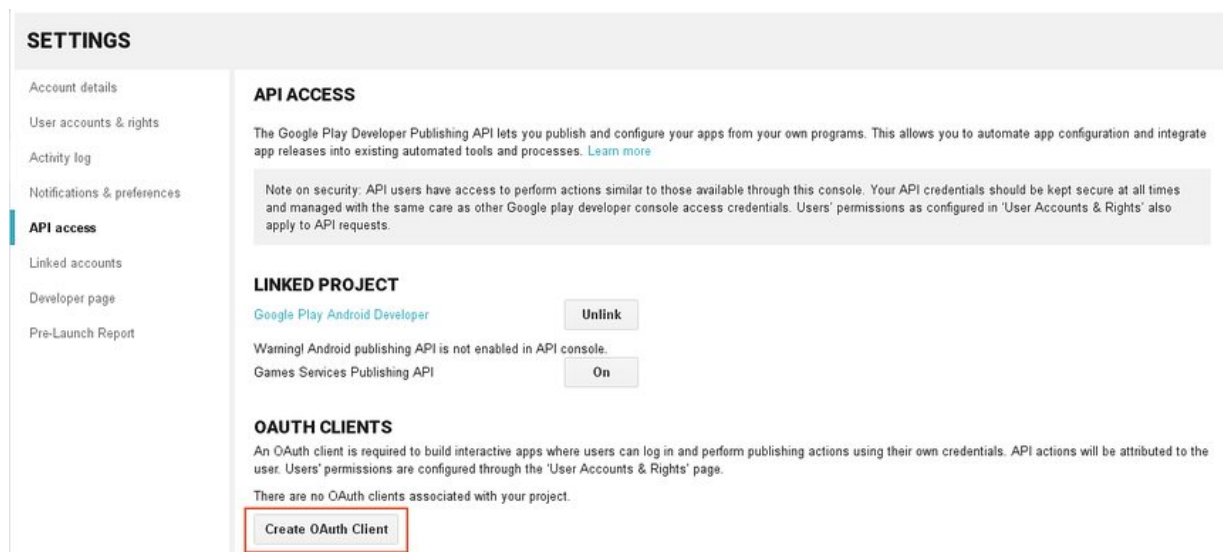


In the **SETTINGS** page, select **API access** and click the **Create new project** button:



After a minute or so, the new API project is automatically generated and linked to your Google Play Developer Console account.

The next step is to create an OAuth Client for the app (if one has not already been created). When users request access to their private data using your app, your OAuth Client ID is used to authenticate your app. Click **Create OAuth Client** to create a new OAuth client:



After a few seconds, a new Client ID is generated. Click **View in Google Developers Console** to see your

new Client ID in the Google Developer's Console:

LINKED PROJECT

Google Play Android Developer

Unlink

Games Services Publishing API

On

OAUTH CLIENTS

An OAuth client is required to build interactive apps where users can log in and perform publishing actions using their own credentials. API actions will be attributed to the user. Users' permissions are configured through the 'User Accounts & Rights' page.

CLIENT ID

65179385121-t15sahafgl0c6roajduf7s.apps.googleusercontent.com

CLIENT SECRET

[View in Google Developers Console](#)

Create OAuth Client

The Client ID is displayed along its name and creation date. Click the **Edit OAuth Client** icon to view the Client secret for your app:

The screenshot shows the Google API Manager interface. On the left is a sidebar with 'API Manager' and a menu containing 'Dashboard', 'Library', and 'Credentials' (which is selected). The main area is titled 'Credentials' and has tabs for 'Credentials', 'OAuth consent screen', and 'Domain verification'. Below the tabs are buttons for 'Create credentials' and 'Delete'. A message states: 'Create credentials to access your enabled APIs. Refer to the API documentation for details.' Below this is a table of 'OAuth 2.0 client IDs'. The table has columns: Name, Creation date, Type, and Client ID. One client is listed: 'Google Play Android Developer' with a creation date of 'Sep 12, 2016' and type 'Other'. The Client ID is '65179385121-t15sahafgl0c6roajduf7s.apps.googleusercontent.com'. To the right of the table is an edit icon (pencil) which is highlighted with a red box.

The default name of the OAuth client is *Google Play Android Developer*. This can be changed to the name of Xamarin.Android app, or any suitable name. In this example, the OAuth Client name is changed to the name of the app, **MyApp**:

This screenshot shows the 'Details' view of an OAuth 2.0 client ID in the Google API Manager. The sidebar is the same as the previous screenshot. The main area shows the 'Client ID for Other' section. At the top are buttons: 'Download JSON', 'Reset secret', and 'Delete'. Below these are the client details: 'Client ID' (65179385121-t15sahafgl0c6roajduf7s.apps.googleusercontent.com), 'Client secret' (BWwsMNI0MbJtIBUjuN6Q), and 'Creation date' (Sep 12, 2016, 2:06:19 PM). The 'Name' field is at the bottom, containing 'MyApp' (highlighted with a red box). 'Save' and 'Cancel' buttons are at the very bottom.

Click **Save** to save changes. This returns to the **Credentials** page where to download the credentials by clicking on the **Download JSON** icon :

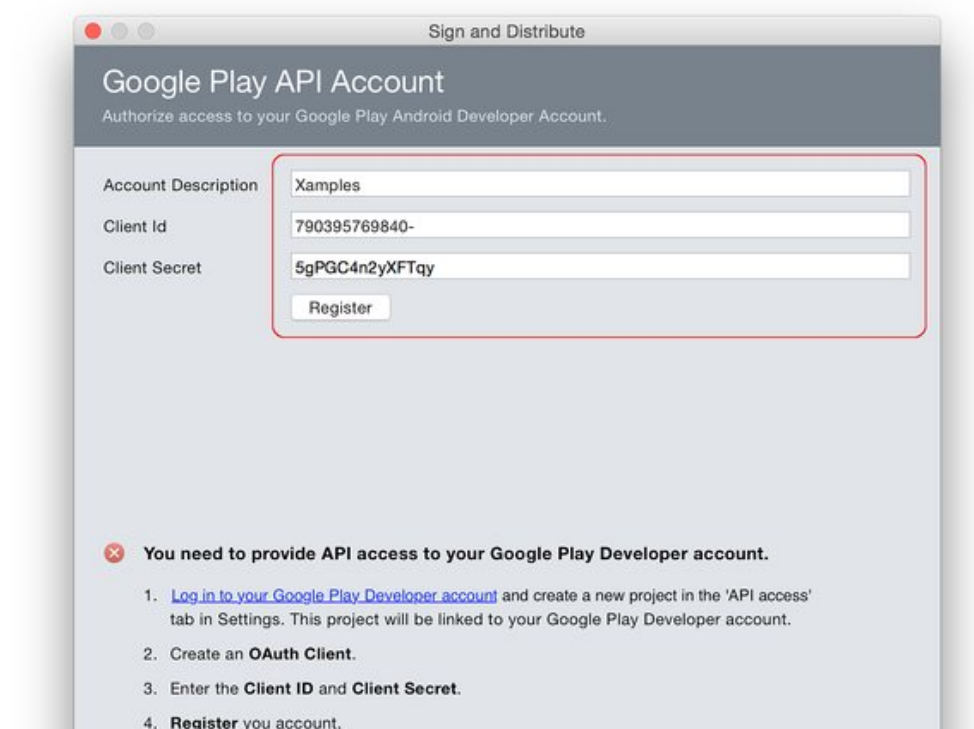


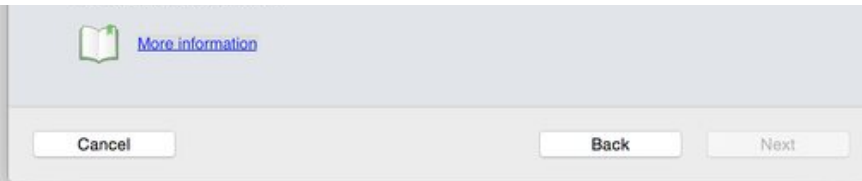
This JSON file contains the Client ID and Client secret that you can cut and paste into the **Sign and Distribute** dialog in the next step.

Register Google API Access

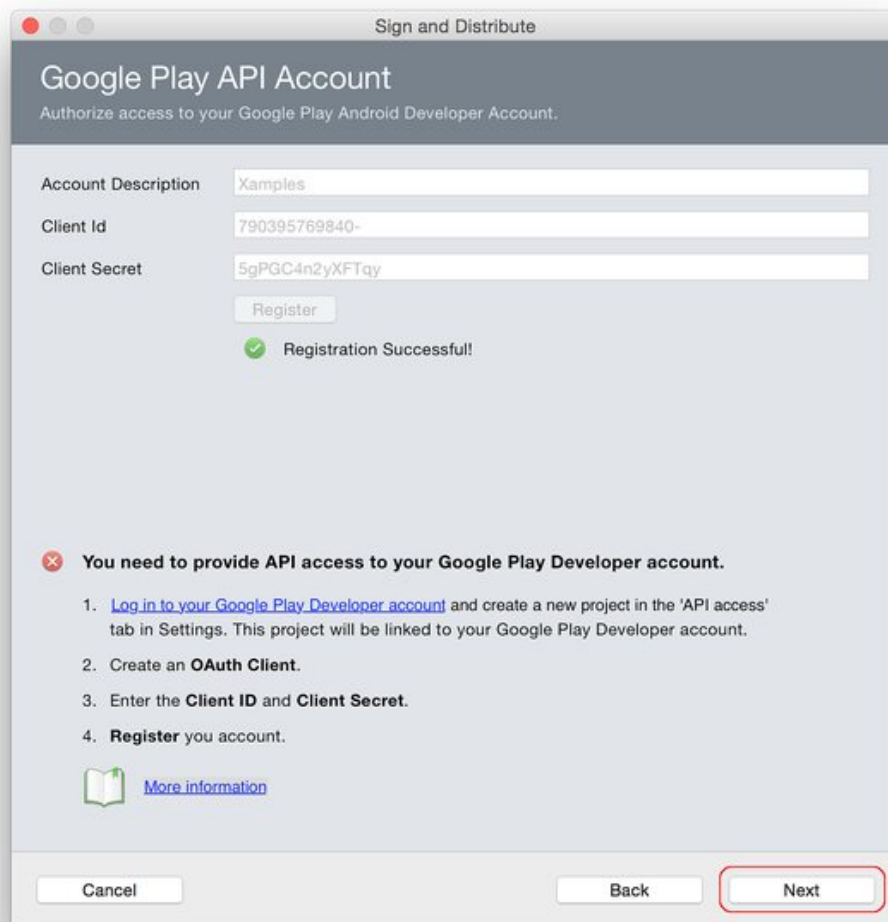
[[ide name="xs"]]

Use the Client ID and Client secret to complete the **Google Play API Account** dialog in Xamarin Studio. It is possible to give the account a description – this makes it possible to register more than one Google Play account and upload future APK's to different Google Play accounts. Copy the Client ID and Client secret to this dialog and click **Register**:



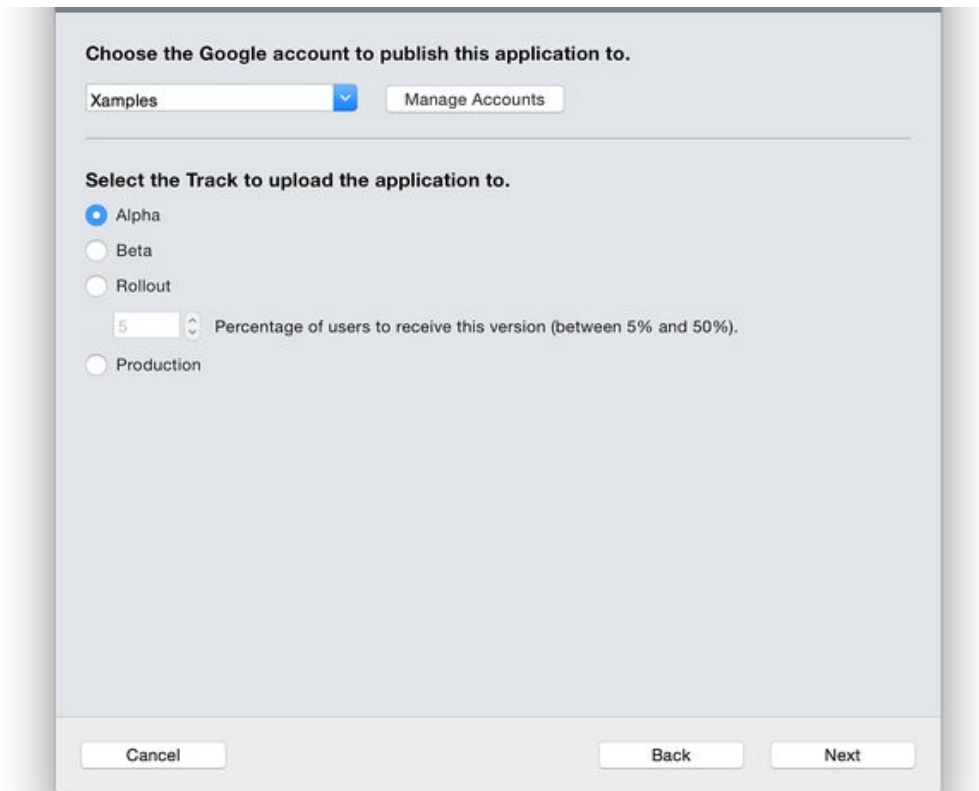


If the Client ID and Client secret are accepted, a **Registration Successful** message is displayed. Click **Next**:



In the **Google Play Account** dialog, select a Google account and a track for uploading the application:





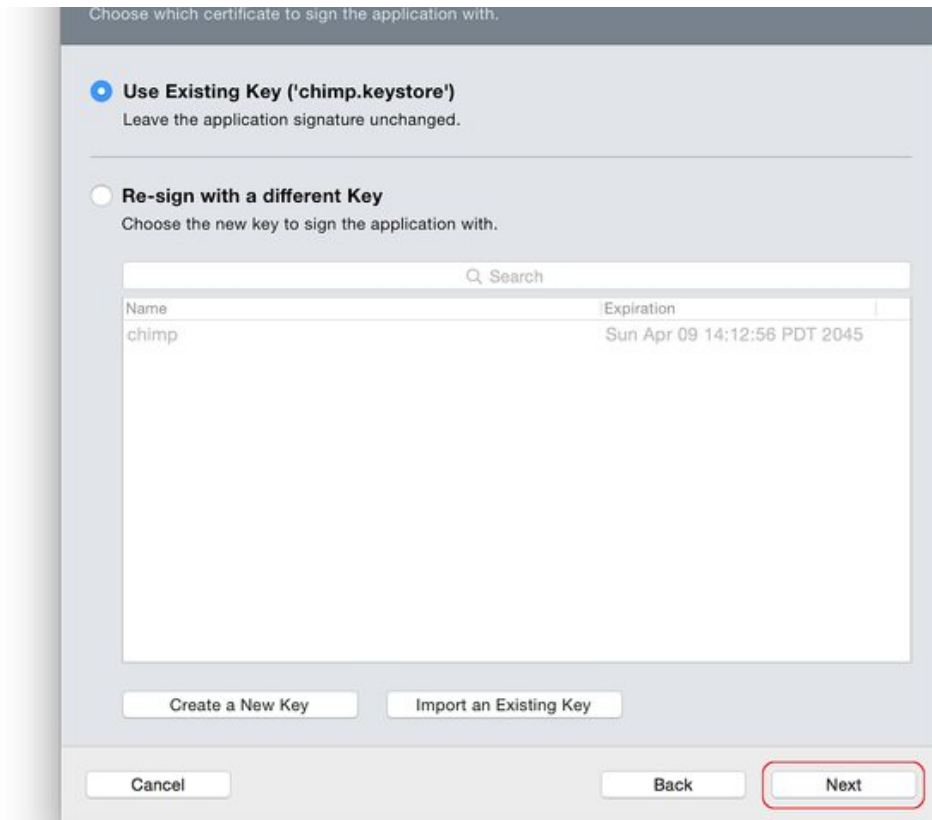
Google Play offers four possible tracks for uploading your app:

- **Alpha** – Used for uploading a very early version of the app to a small list of testers.
- **Beta** – Used for uploading an early version of the app to a larger list of testers.
- **Rollout** – Allows a percentage of users to receive an updated version of the app; this makes it possible to slowly increase the percentage from say, 10% of users and increase it to 100% of users while you iron out bugs.
- **Production** – Select this track when the app is ready for full distribution from the Google Play store.

For more information about Google Play testing and staged rollouts, see [Set up alpha/beta tests](#).

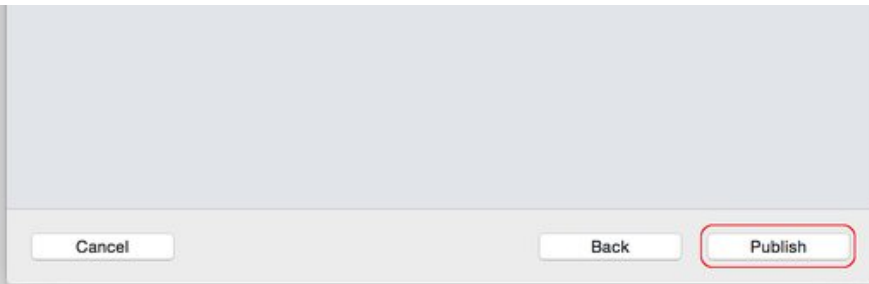
Next, choose a signing identity to that will be used to sign the app. Select **Use Existing Key** in order to use an existing signing identity, otherwise consult the guide [Creating a New Certificate](#) for information about creating a new key. After you have selected a certificate to sign the application, click **Next**:





At this point the app can be uploaded to Google Play. The **Publish to Google Play** dialog summarizes information about your app – click **Publish** to publish your app to Google Play:





Note that one APK must have already been submitted to the Google Play store before the **Publish to Google Play** will work. If an APK is not uploaded the following error may occur:

Google Play requires you to manually upload your first APK for this app. You can use an ad-hoc APK for this.

or

No application was found for the given package name. [404]

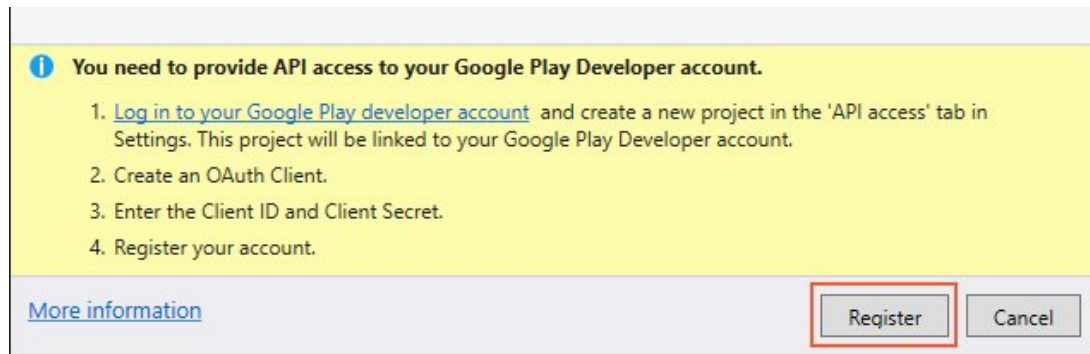
To resolve this error, manually upload an APK (such as an Ad-Hoc build) via the Google Play Developer Console and use the **Publish to Google Play** dialog for subsequent APK updates. For information about how to manually upload an APK, see [Manually Uploading the APK](#).

[[/ide]]

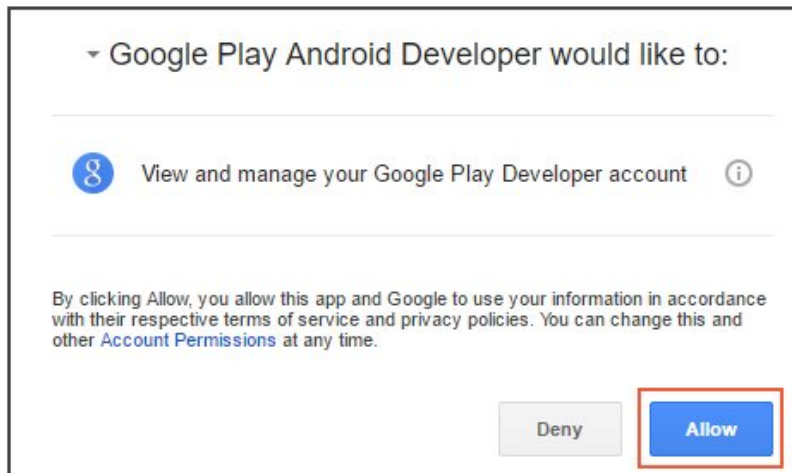
[[ide name="vs"]]

Use the Client ID and Client secret to complete the **Google Play API Account** dialog in Xamarin Studio. It is possible to give the account a description – this makes it possible to register more than one Google Play account and upload future APK's to different Google Play accounts. Copy the Client ID and Client secret to this dialog and click **Register**:

A screenshot of a dialog box titled 'Register Google API Access'. It contains three text input fields. The first field is labeled 'Account Description:' and contains the text 'MyApp'. The second field is labeled 'Client Id:' and contains the text '65179385121-t15sahafgl0c6roajduf7s.apps.googleusercontent.com'. The third field is labeled 'Client Secret:' and contains the text 'BWwsMNIOMbJtIBUjuN6Q'. The dialog has a close button (X) in the top right corner.

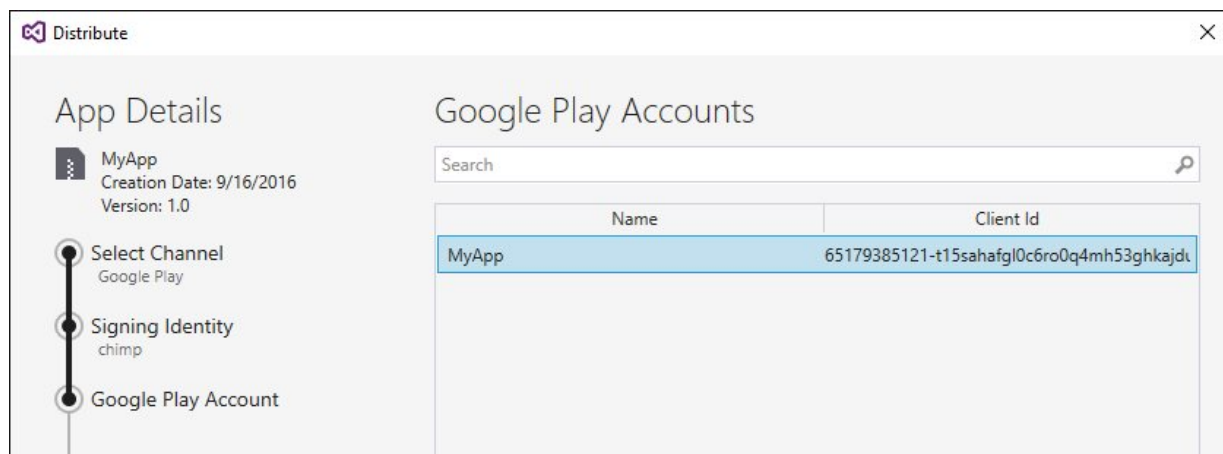


A web browser will open and prompt you to sign into your Google Play Android Developer account (if you are not already signed in). After you sign in, the following prompt is displayed in the web browser. Click **Allow** to authorize the app:



Publish

After clicking **Allow**, the browser reports *Received verification code. Closing...* and the app is added to the list of Google Play Accounts in Visual Studio. In the **Google Play Accounts** dialog, click **Continue**:

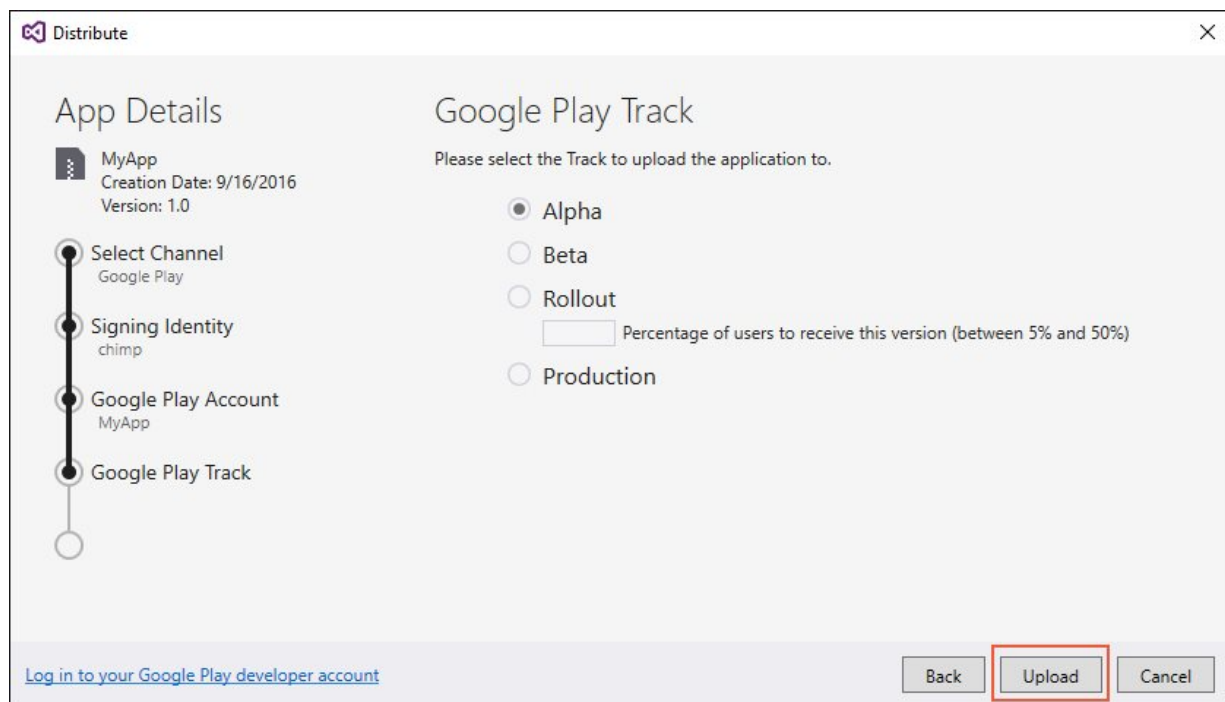




Next, the **Google Play Track** dialog is presented. Google Play offers four possible tracks for uploading your app:

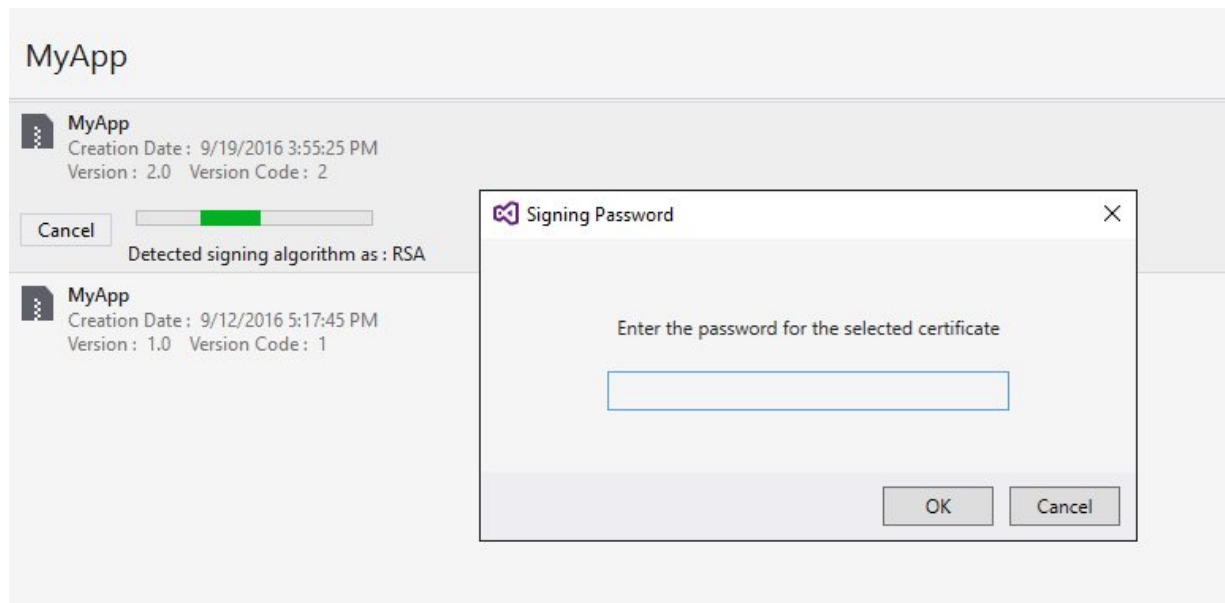
- **Alpha** – Used for uploading a very early version of the app to a small list of testers.
- **Beta** – Used for uploading an early version of the app to a larger list of testers.
- **Rollout** – Allows a percentage of users to receive an updated version of the app; this makes it possible to slowly increase the percentage from say, 10% of users and increase it to 100% of users while you iron out bugs.
- **Production** – Select this track when the app is ready for full distribution from the Google Play store.

Choose which Google Play track will be used for uploading the app and click **Upload**. If you select **Rollout**, be sure to enter a percentage value:

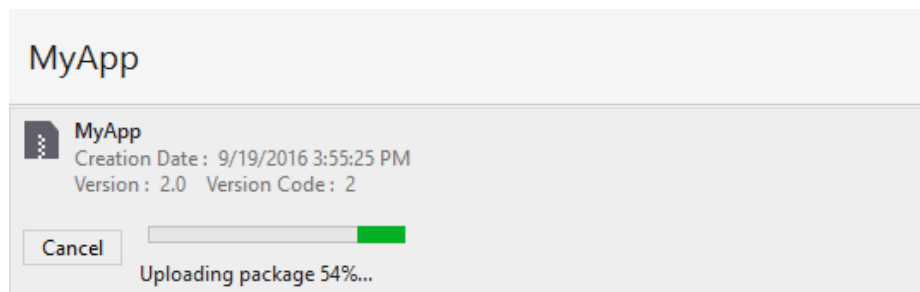


For more information about Google Play testing and staged rollouts, see [Set up alpha/beta tests](#).

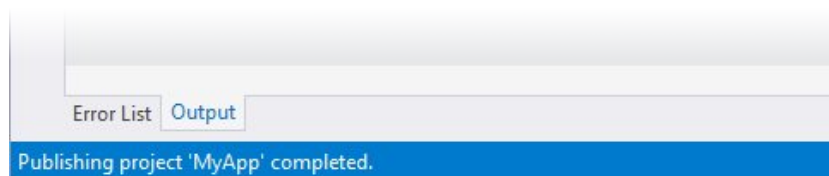
Next, a dialog is presented to enter the password for the signing certificate. Enter the password and click **OK**:



The **Archive Manager** displays the progress of the upload:

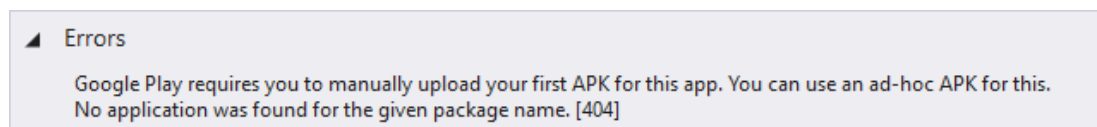


When the upload finishes, completion status is shown in the lower left hand corner of Visual Studio:



Troubleshooting

Note that one APK must have already been submitted to the Google Play store before the **Publish to Google Play** will work. If an APK is not already uploaded the Publishing Wizard will display the following error in the **Errors** pane:



[Goto Google Play...](#)

When this error occurs, manually upload an APK (such as an Ad-Hoc build) via the Google Play Developer Console and use the **Distribution Channel** dialog for subsequent APK updates. For more information, see [Manually Uploading the APK](#). The version code of the APK must change with each upload, otherwise the following error will occur:

▲ Errors

A APK with version code (1) has already been uploaded.

To resolve this error, rebuild the app with a different version number and resubmit it to Google Play via the **Distribution Channel** dialog.

[[/ide]]