

Understanding Android API Levels

📅 06/01/2018 ⌚ 18 minutes to read Contributors 

In this article

[Quick Start](#)

[Android Versions and API Levels](#)

[Project API Level Settings](#)

[Runtime Checks for Android Versions](#)

[API Levels and Libraries](#)

[Summary](#)

[Related Links](#)

Xamarin.Android has several Android API level settings that determine your app's compatibility with multiple versions of Android. This guide explains what these settings mean, how to configure them, and what effect they have on your app at run time.

Quick Start

Xamarin.Android exposes three Android API level project settings:

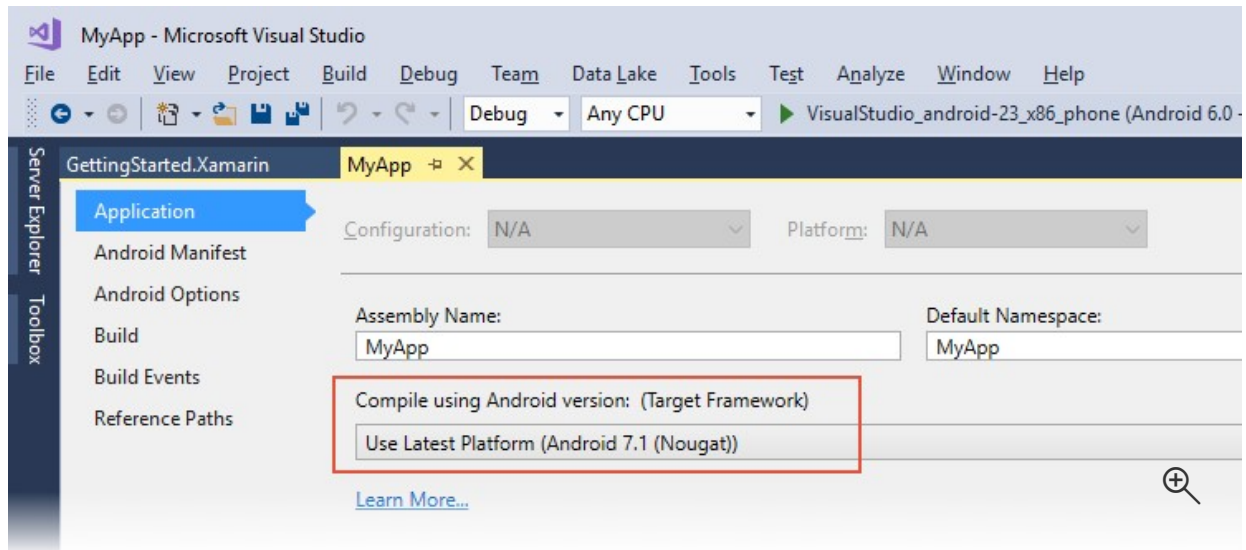
- [Target Framework](#) – Specifies which framework to use in building your application. This API level is used at *compile* time by Xamarin.Android.
- [Minimum Android Version](#) – Specifies the oldest Android version that you want your app to support. This API level is used at *run* time by Android.
- [Target Android Version](#) – Specifies the version of Android that your app is intended to run on. This API level is used at *run* time by Android.

Before you can configure an API level for your project, you must install the SDK platform components for that API level. For more information about downloading and installing Android SDK components, see [Android SDK Setup](#).

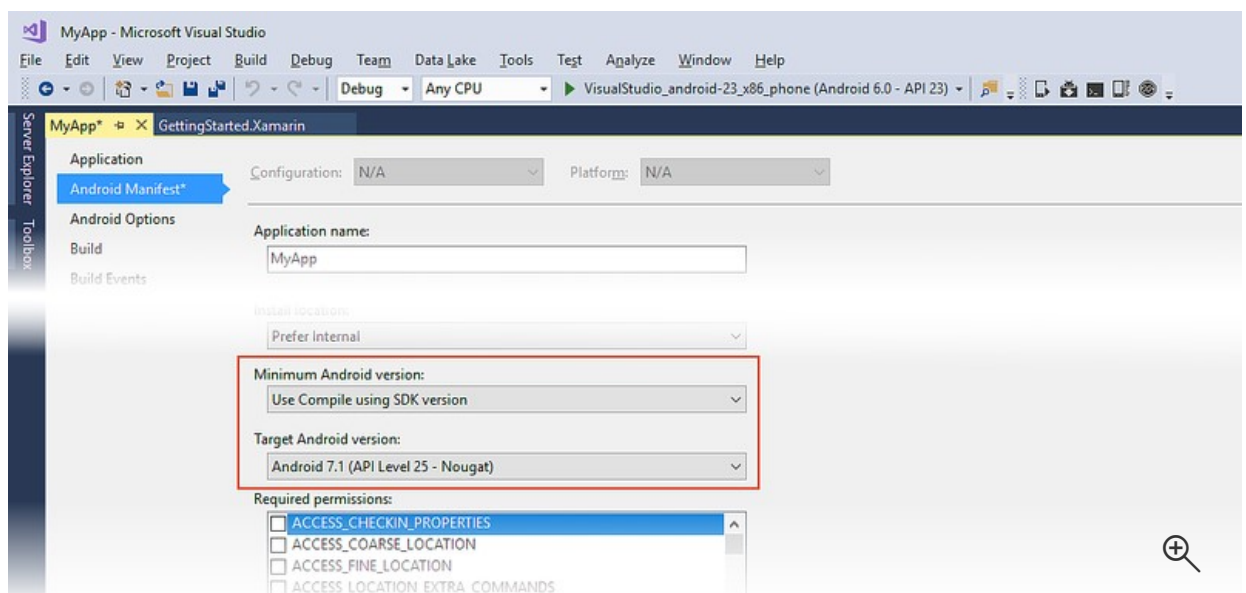
Note

Beginning in August 2018, the Google Play Console will require that new apps target API level 26 (Android 8.0) or higher. Existing apps will be required to target API level 26 or higher beginning in November 2018. For more information, see [Improving app security and performance on Google Play for years to come](#).

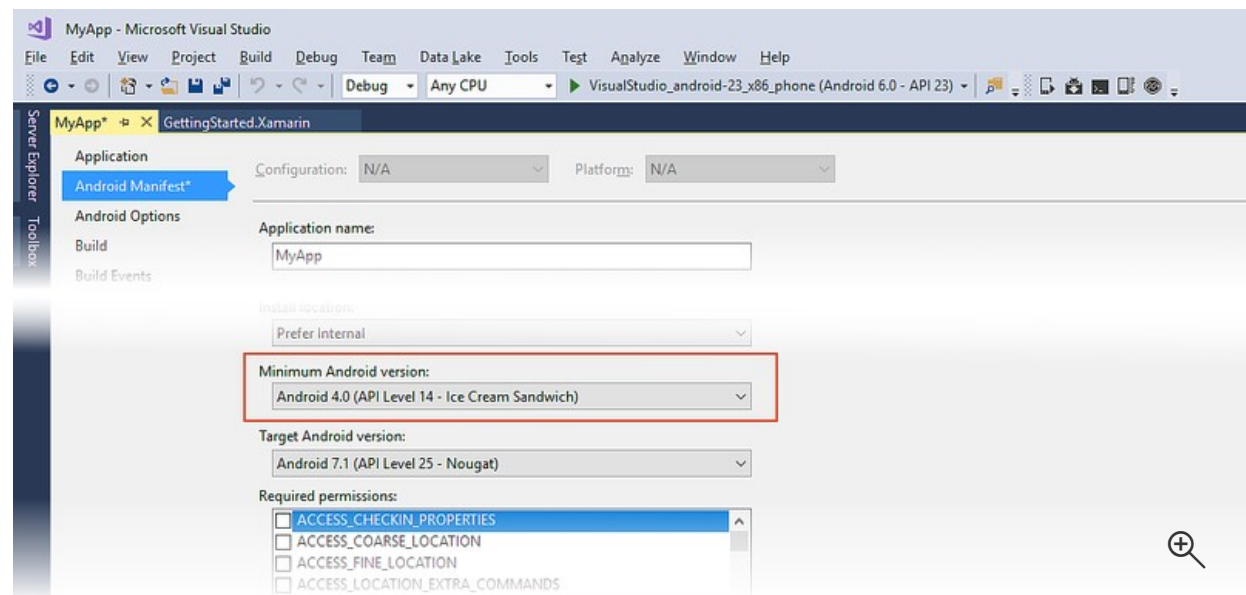
Normally, all three Xamarin.Android API levels are set to the same value. On the **Application** page, set **Compile using Android version (Target Framework)** to the latest stable API version (or, at a minimum, to the Android version that has all of the features you need). In the following screenshot, the Target Framework is set to **Android 7.1 (API Level 25 - Nougat)**:



On the **Android Manifest** page, set the Minimum Android version to **Use Compile using SDK version** and set the Target Android version to the same value as the Target Framework version (in the following screenshot, the Target Android Framework is set to **Android 7.1 (Nougat)**):



If you want to maintain backward compatibility with an earlier version of Android, set **Minimum Android version to target** to the oldest version of Android that you want your app to support. (Note that API Level 14 is the minimum API level required for [Google Play services and Firebase support](#).) The following example configuration supports Android versions from API Level 14 thru API level 25:



If your app supports multiple Android versions, your code must include runtime checks to ensure that your app works with the Minimum Android version setting (see [Runtime Checks for Android Versions](#) below for details). If you are consuming or creating a library, see [API Levels and Libraries](#) below for best practices in configuring API level settings for libraries.

Android Versions and API Levels

As the Android platform evolves and new Android versions are released, each Android version is assigned a unique integer identifier, called the *API Level*. Therefore, each Android version corresponds to a single Android API Level. Because users install apps on older as well as the most recent versions of Android, real-world Android apps must be designed to work with multiple Android API levels.

Android Versions

Each release of Android goes by multiple names:

- The Android version, such as **Android 7.1**
- A code name, such as *Nougat*

- A corresponding API level, such as **API level 25**

An Android code name may correspond to multiple versions and API levels (as seen in the list below), but each Android version corresponds to exactly one API level.

In addition, Xamarin.Android defines *build version codes* that map to the currently known Android API levels. The following list can help you translate between API level, Android version, code name, and Xamarin.Android build version code.

- **API 27 (Android 8.1)** – *Oreo*, released December 2017. Build version code

```
Android.OS.BuildVersionCodes.OMr1
```

- **API 26 (Android 8.0)** – *Oreo*, released August 2017. Build version code

```
Android.OS.BuildVersionCodes.O
```

- **API 25 (Android 7.1)** – *Nougat*, released December 2016. Build version code

```
Android.OS.BuildVersionCodes.NMr1
```

- **API 24 (Android 7.0)** – *Nougat*, released August 2016. Build version code

```
Android.OS.BuildVersionCodes.N
```

- **API 23 (Android 6.0)** – *Marshmallow*, released August 2015. Build version code

```
Android.OS.BuildVersionCodes.M
```

- **API 22 (Android 5.1)** – *Lollipop*, released March 2015. Build version code

```
Android.OS.BuildVersionCodes.LollipopMr1
```

- **API 21 (Android 5.0)** – *Lollipop*, released November 2014. Build version code

```
Android.OS.BuildVersionCodes.Lollipop
```

- **API 20 (Android 4.4W)** – *Kitkat Watch*, released June 2014. Build version code

```
Android.OS.BuildVersionCodes.KitKatWatch
```

- **API 19 (Android 4.4)** – *Kitkat*, released October 2013. Build version code

```
Android.OS.BuildVersionCodes.KitKat
```

- **API 18 (Android 4.3)** – *Jelly Bean*, released July 2013. Build version code

```
Android.OS.BuildVersionCodes.JellyBeanMr2
```

- **API 17 (Android 4.2-4.2.2)** – *Jelly Bean*, released November 2012. Build version code

```
Android.OS.BuildVersionCodes.JellyBeanMr1
```

- **API 16 (Android 4.1-4.1.1)** – *Jelly Bean*, released June 2012. Build version code
`Android.OS.BuildVersionCodes.JellyBean`
- **API 15 (Android 4.0.3-4.0.4)** – *Ice Cream Sandwich*, released December 2011. Build version code
`Android.OS.BuildVersionCodes.IceCreamSandwichMr1`
- **API 14 (Android 4.0-4.0.2)** – *Ice Cream Sandwich*, released October 2011. Build version code
`Android.OS.BuildVersionCodes.IceCreamSandwich`
- **API 13 (Android 3.2)** – *Honeycomb*, released June 2011. Build version code
`Android.OS.BuildVersionCodes.HoneyCombMr2`
- **API 12 (Android 3.1.x)** – *Honeycomb*, released May 2011. Build version code
`Android.OS.BuildVersionCodes.HoneyCombMr1`
- **API 11 (Android 3.0.x)** – *Honeycomb*, released February 2011. Build version code
`Android.OS.BuildVersionCodes.HoneyComb`
- **API 10 (Android 2.3.3-2.3.4)** – *Gingerbread*, released February 2011. Build version code
`Android.OS.BuildVersionCodes.GingerBreadMr1`
- **API 9 (Android 2.3-2.3.2)** – *Gingerbread*, released November 2010. Build version code
`Android.OS.BuildVersionCodes.GingerBread`
- **API 8 (Android 2.2.x)** – *Froyo*, released June 2010. Build version code
`Android.OS.BuildVersionCodes.Froyo`
- **API 7 (Android 2.1.x)** – *Eclair*, released January 2010. Build version code
`Android.OS.BuildVersionCodes.EclairMr1`
- **API 6 (Android 2.0.1)** – *Eclair*, released December 2009. Build version code
`Android.OS.BuildVersionCodes.Eclair01`
- **API 5 (Android 2.0)** – *Eclair*, released November 2009. Build version code
`Android.OS.BuildVersionCodes.Eclair`
- **API 4 (Android 1.6)** – *Donut*, released September 2009. Build version code
`Android.OS.BuildVersionCodes.Donut`
- **API 3 (Android 1.5)** – *Cupcake*, released May 2009. Build version code
`Android.OS.BuildVersionCodes.Cupcake`

- **API 2 (Android 1.1)** – *Base*, released February 2009. Build version code

```
Android.OS.BuildVersionCodes.Base11
```

- **API 1 (Android 1.0)** – *Base*, released October 2008. Build version code

```
Android.OS.BuildVersionCodes.Base
```

As this list indicates, new Android versions are released frequently – sometimes several releases per year. As a result, the universe of Android devices that might run your app includes of a wide variety of older and newer Android versions. How can you guarantee that your app will run consistently and reliably on so many different versions of Android? Android's API levels can help you manage this problem.

Android API Levels

Each Android device runs at exactly *one* API level – this API level is guaranteed to be unique per Android platform version. The API level precisely identifies the version of the API set that your app can call into; it identifies the combination of manifest elements, permissions, etc. that you code against as a developer. Android's system of API levels helps Android determine whether an application is compatible with an Android system image prior to installing the application on a device.

When an application is built, it contains the following API level information:

- The *target* API level of Android that the app is built to run on.
- The *minimum* Android API level that an Android device must have to run your app.

These settings are used to ensure that the functionality needed to run the app correctly is available on the Android device at installation time. If not, the app is blocked from running on that device. For example, if the API level of an Android device is lower than the minimum API level that you specify for your app, the Android device will prevent the user from installing your app.

Project API Level Settings

The following sections explain how to use the SDK Manager to prepare your development environment for the API levels you want to target, followed by detailed explanations of how to configure *Target Framework*, *Minimum Android version*, and *Target Android version* settings in Xamarin.Android.

Android SDK Platforms

Before you can select a Target or Minimum API level in Xamarin.Android, you must install the Android SDK platform version that corresponds to that API level. The range of available choices for Target Framework, Minimum Android version, and Target Android version is limited to the range of Android SDK versions that you have installed. You can use the SDK Manager to verify that the required Android SDK versions are installed, and you can use it to add any new API levels that you need for your app. If you are not familiar with how to install API levels, see [Android SDK Setup](#).

Target Framework

The *Target Framework* (also known as `compileSdkVersion`) is the specific Android framework version (API level) that your app is compiled for at build time. This setting specifies what APIs your app *expects* to use when it runs, but it has no effect on which APIs are actually available to your app when it is installed. As a result, changing the Target Framework setting does not change runtime behavior.

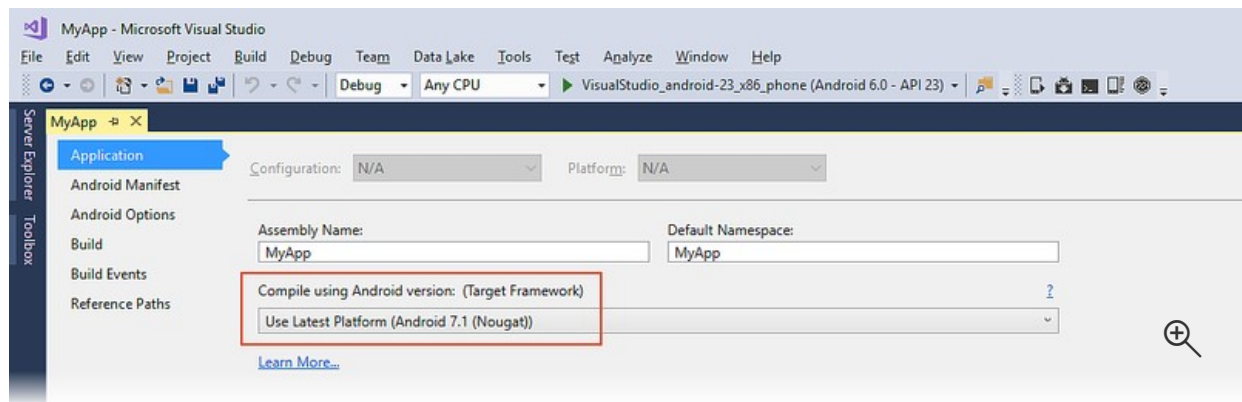
The Target Framework identifies which library versions your application is linked against – this determines which APIs you can use in your app. For example, if you want to use the [NotificationBuilder.SetCategory](#) method that was introduced in Android 5.0 Lollipop, you must set the Target Framework to **API Level 21 (Lollipop)** or later. If you set your project's Target Framework to an API level such as **API Level 19 (KitKat)** and try to call the `SetCategory` method in your code, you will get a compile error.

We recommend that you always compile with the *latest* available Target Framework version. Doing so provides you with helpful warning messages for any deprecated APIs that might be called by your code. Using the latest Target Framework version is especially important when you use the latest support library releases – each library expects your app to be compiled at that support library's minimum API level or greater.

Visual Studio

Visual Studio for Mac

To access the Target Framework setting in Visual Studio, open the project properties in **Solution Explorer** and select the **Application** page:



Set the Target Framework by selecting an API level in the drop-down menu under **Compile using Android version** as shown above.

Minimum Android Version

The *Minimum Android version* (also known as `minSdkVersion`) is the oldest version of the Android OS (i.e., the lowest API level) that can install and run your application. By default, an app can only be installed on devices matching the Target Framework setting or higher; if the Minimum Android version setting is *lower* than the Target Framework setting, your app can also run on earlier versions of Android. For example, if you set the Target Framework to **Android 7.1 (Nougat)** and set the Minimum Android version to **Android 4.0.3 (Ice Cream Sandwich)**, your app can be installed on any platform from API level 15 to API level 25, inclusive.

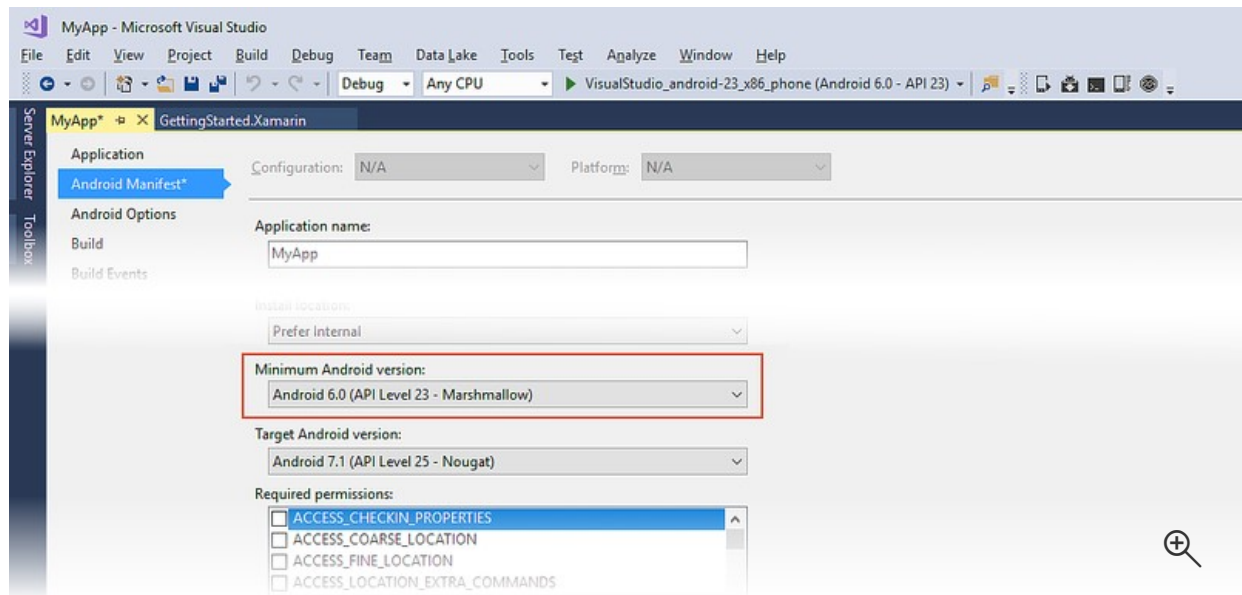
Although your app may successfully build and install on this range of platforms, this does not guarantee that it will successfully *run* on all of these platforms. For example, if your app is installed on **Android 5.0 (Lollipop)** and your code calls an API that is available only in **Android 7.1 (Nougat)** and newer, your app will get a runtime error and possibly crash. Therefore, your code must ensure – at runtime – that it calls only those APIs that are supported by the Android device that it is running on. In other words, your code must include explicit runtime checks to ensure that your app uses newer APIs only on devices that are recent enough to support them. [Runtime Checks for Android Versions](#), later in this guide, explains how to add these runtime checks to your code.

Visual Studio

Visual Studio for Mac

To access the Minimum Android version setting in Visual Studio, open the project properties in **Solution Explorer** and select the **Android Manifest** page. In the drop-

down menu under **Minimum Android version** you can select the Minimum Android version for your application:



If you select **Use Compile using SDK version**, the Minimum Android version will be the same as the Target Framework setting.

Target Android Version

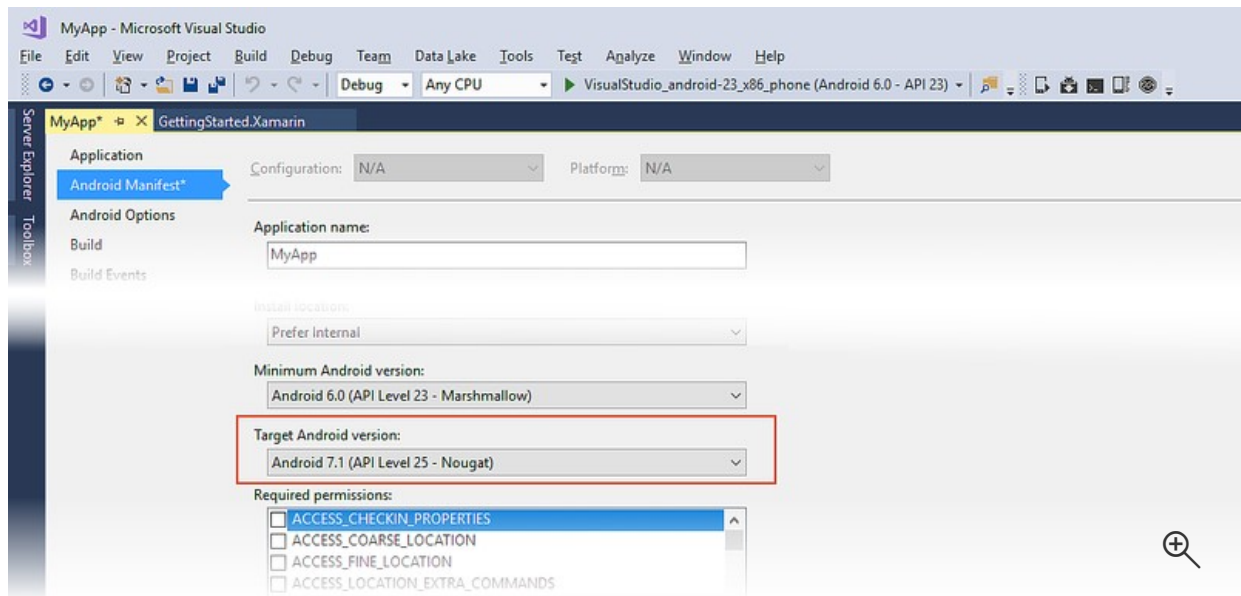
The *Target Android Version* (also known as `targetSdkVersion`) is the API level of the Android device where the app expects to run. Android uses this setting to determine whether to enable any compatibility behaviors – this ensures that your app continues to work the way you expect. Android uses the Target Android version setting of your app to figure out which behavior changes can be applied to your app without breaking it (this is how Android provides forward compatibility).

The Target Framework and the Target Android version, while having very similar names, are not the same thing. The Target Framework setting communicates target API level information to Xamarin.Android for use at *compile time*, while the Target Android version communicates target API level information to Android for use at *run time* (when the app is installed and running on a device).

Visual Studio

Visual Studio for Mac

To access this setting in Visual Studio, open the project properties in **Solution Explorer** and select the **Android Manifest** page. In the drop-down menu under **Target Android version** you can select the Target Android version for your application:



We recommend that you explicitly set the Target Android version to the latest version of Android that you use to test your app. Ideally, it should be set to the latest Android SDK version – this allows you to use new APIs prior to working through the behavior changes. For most developers, we *do not* recommend setting the Target Android version to **Use Compile using SDK version**.

In general, the Target Android Version should be bounded by the Minimum Android Version and the Target Framework. That is:

Minimum Android Version <= Target Android Version <= Target Framework

For more information about SDK levels, see the Android Developer [uses-sdk](#) documentation.


Runtime Checks for Android Versions

As each new version of Android is released, the framework API is updated to provide new or replacement functionality. With few exceptions, API functionality from earlier Android versions is carried forward into newer Android versions without modifications. As a result, if your app runs on a particular Android API level, it will typically be able to run on a later Android API level without modifications. But what if you also want to run your app on earlier versions of Android?

If you select a Minimum Android version that is *lower* than your Target Framework setting, some APIs may not be available to your app at runtime. However, your app can still run on an earlier device, but with reduced functionality. For each API that is not available on


Android platforms corresponding to your Minimum Android version setting, your code must explicitly check the value of the `Android.OS.Build.VERSION.SdkInt` property to determine the API level of the platform the app is running on. If the API level is *lower* than the Minimum Android version that supports the API you want to call, then your code has to find a way to function properly without making this API call.

For example, let's suppose that we want to use the [NotificationBuilder.SetCategory](#) method to categorize a notification when running on **Android 5.0 Lollipop** (and later), but we still want our app to run on earlier versions of Android such as **Android 4.1 Jelly Bean** (where `SetCategory` is not available). Referring to the Android version table at the beginning of this guide, we see that the build version code for **Android 5.0 Lollipop** is `Android.OS.BuildVersionCodes.Lollipop`. To support older versions of Android where `SetCategory` is not available, our code can detect the API level at runtime and conditionally call `SetCategory` only when the API level is greater than or equal to the Lollipop build version code:

C#	
<pre>if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop) { builder.SetCategory(Notification.CategoryEmail); }</pre>	

In this example, our app's Target Framework is set to **Android 5.0 (API Level 21)** and its Minimum Android version is set to **Android 4.1 (API Level 16)**. Because `SetCategory` is available in API level `Android.OS.BuildVersionCodes.Lollipop` and later, this example code will call `SetCategory` only when it is actually available – it will *not* attempt to call `SetCategory` when the API level is 16, 17, 18, 19, or 20. The functionality is reduced on these earlier Android versions only to the extent that notifications are not sorted properly (because they are not categorized by type), yet the notifications are still published to alert the user. Our app still works, but its functionality is slightly diminished.

In general, the build version check helps your code decide at runtime between doing something the new way versus the old way. For example:

C#	
<pre>if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.Lollipop) { // Do things the Lollipop way }</pre>	

```

}
else
{
    // Do things the pre-Lollipop way
}

```

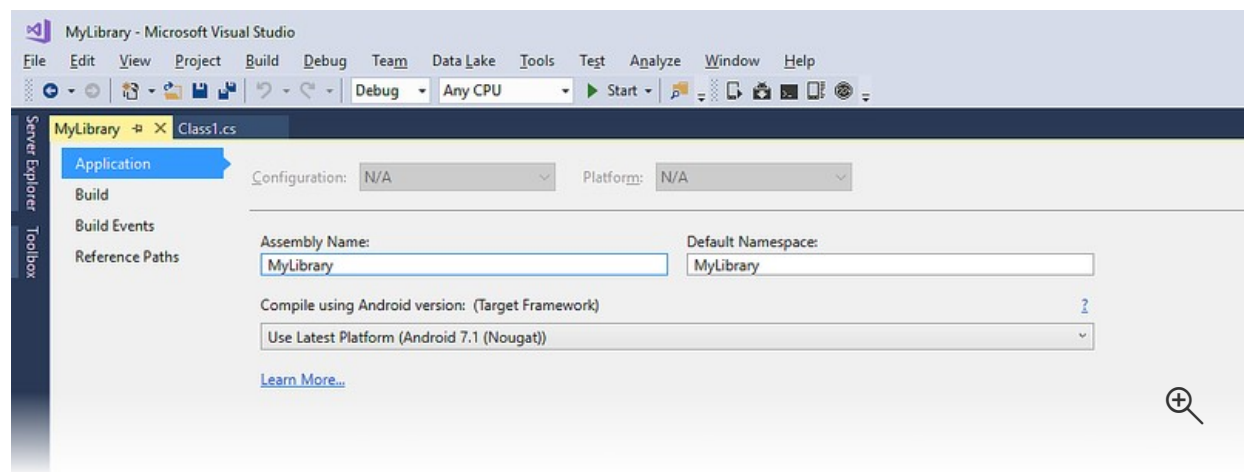
There's no fast and simple rule that explains how to reduce or modify your app's functionality when it runs on older Android versions that are lacking one or more APIs. In some cases (such as in the `SetCategory` example above), it's sufficient to simply omit the API call when it's not available. However, in other cases, you may need to implement alternate functionality for when `Android.OS.Build.VERSION.SdkInt` is detected to be less than the API level that your app needs to present its optimum experience.

API Levels and Libraries

Visual Studio

Visual Studio for Mac

When you create a Xamarin.Android library project (such as a class library or a bindings library), you can configure only the Target Framework setting – the Minimum Android version and the Target Android version settings are not available. That is because there is no **Android Manifest** page:



The Minimum Android version and Target Android version settings are not available because the resulting library is not a stand-alone app – the library could be run on any Android version, depending on the app that it is packaged with. You can specify how the library is to be *compiled*, but you can't predict which platform API level the library will be run on. With this in mind, the following best practices should be observed when consuming or creating libraries:

- **When consuming an Android library** – If you are consuming an Android library in your application, be sure to set your app's Target Framework setting to an API level that is *at least as high as* the Target Framework setting of the library.
- **When creating an Android library** – If you are creating an Android library for use by other applications, be sure to set its Target Framework setting to the minimum API level that it needs in order to compile.

These best practices are recommended to help prevent the situation where a library attempts to call an API that is not available at runtime (which can cause the app to crash). If you are a library developer, you should strive to restrict your usage of API calls to a small and well-established subset of the total API surface area. Doing so helps to ensure that your library can be used safely across a wider range of Android versions.

Summary

This guide explained how Android API levels are used to manage app compatibility across different versions of Android. It provided detailed steps for configuring the Xamarin.Android *Target Framework*, *Minimum Android version*, and *Target Android version* project settings. It provided instructions for using the Android SDK Manager to install SDK packages, included examples of how to write code to deal with different API levels at runtime, and explained how to manage API levels when creating or consuming Android libraries. It also provided a comprehensive list that relates API levels to Android version numbers (such as Android 4.4), Android version names (such as Kitkat), and Xamarin.Android build version codes.

Related Links

- [Android SDK Setup](#)
- [SDK CLI Tooling Changes](#)
- [Picking your compileSdkVersion, minSdkVersion, and targetSdkVersion](#)
- [What is API Level?](#)
- [Codenames, Tags, and Build Numbers](#)