targetSdkVersion
VS
compileSdkVersion

A Thread

/hellosuman

Both **compileSdkVersion** and **targetSdkVersion** are crucial to handle **forward compatibility** in Android — so they both are connected with what to do when the new Android version appears. But how do they exactly work?

compileSdkVersion defines which Android SDK version will be used by gradle to compile your app.

For **example**:
In Android 12, so in SDK version 31, there was a new API introduced, that allows us to easily implement a splash screen. In this new API, the splash screen can be customized using those properties:

compileSdkVersion 31

activity_main.xml | themes.xml

```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.SerializableParcelable" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <item name="android:windowSplashScreenBackground" tools:targetApi="s">@color/white</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

compileSdkVersion 30 (compile time error)

activity_main.xml | themes.xml

```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.SerializableParcelable" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <item name="android:windowSplashScreenBackground" tools:targetApi="s">@color/white</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Keep in mind

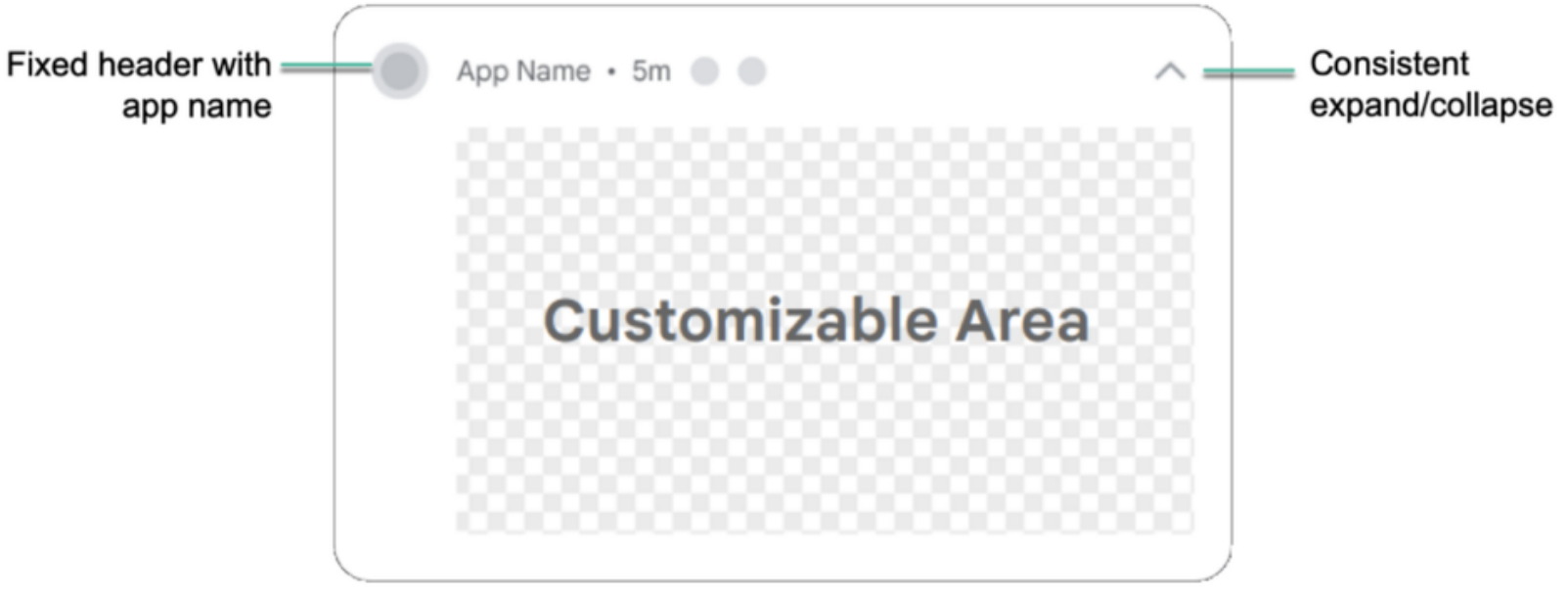
1. That doesn’t of course mean that you can use this new API and forget about users who have older Android versions where this API is not available. If the **minSdkVersion** in your app is lower than 31 you have to also provide an alternative way to display a splash screen for those older devices which do not have access to this new API.
2. Similarly, some methods or properties could be **deprecated** in the **newer** Android SDK version, and some of them even are **removed**. That's why once you increase the **compiledSdkVersion** in your app, you will often see some **warnings** and **errors** during compilation that you have to address.

targetSdkVersion is a property that tells the system up to which Android version the app was designed and tested so that the system should not enable any **compatibility behaviors** to the app up to that version.

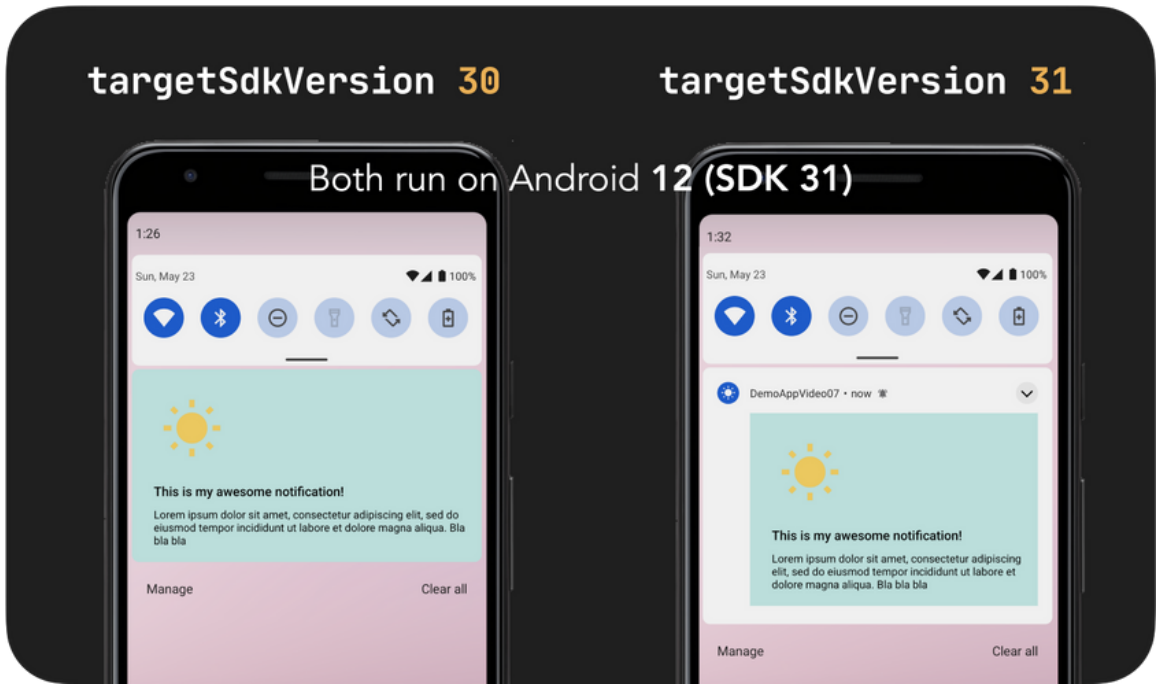
If the API level of the device is higher than the version declared by your app's targetSdkVersion, the system may enable **compatibility behaviors** to ensure that your app continues to work the way you expect.

Example

In Android 12 the appearance of custom notifications was changed. Previously they could use the whole notification area, but in Android 12 system applies the standard template to all custom notifications so they look more consistent.



If your **targetSdkVersion** is below **31,** the system will assume that you haven't tested that feature, so it will add **compatibility behaviors** to display notifications in the old way to minimize the risk that notifications will not be displayed properly. Only after you update the target SDK version to 31 the new notification appearance will be used.

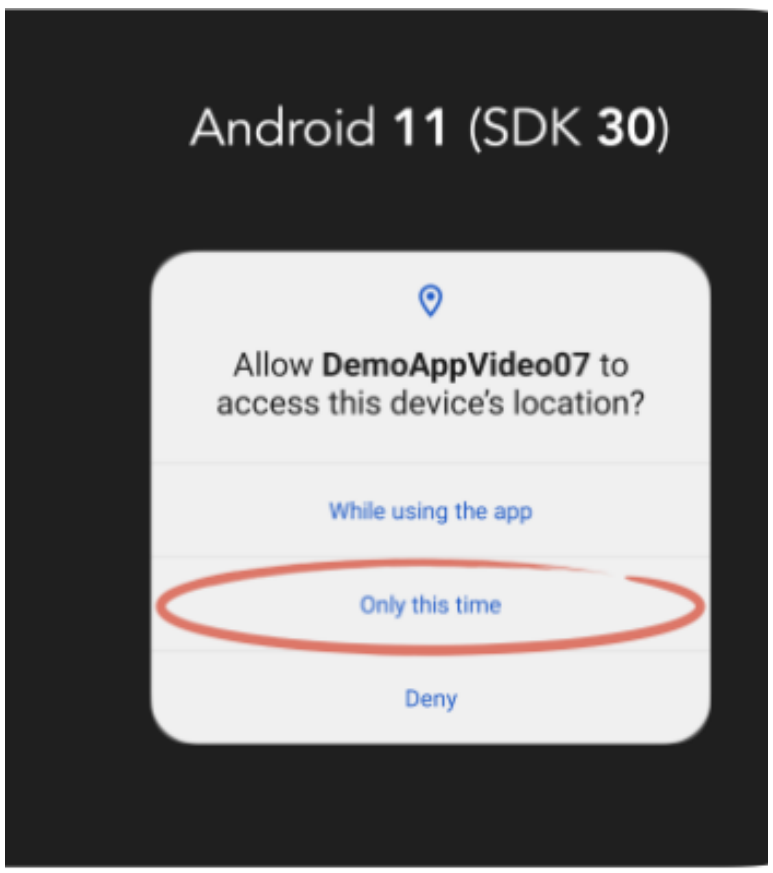


Keep in mind

Not all changes that are introduced in new Android versions use these compatibility mechanisms.

Example:

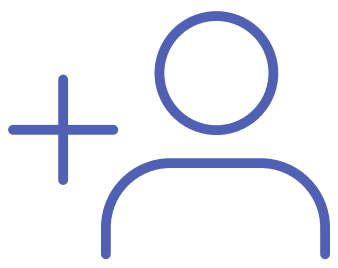
When the device uses Android version 11 or higher and the app asks for location permission, the user can grant temporary access to that data and the app has to handle that case properly no matter if it targets SDK version **below 30** or not.



1. **targetSdkVersion** cannot be higher than the **compileSdkVersion** simply because we cannot target things that we know nothing about during compilation.
2. Ideally, the **compileSdkVersion** and **targetSdkVersion** should be equal and both point to the latest SDK. But of course only after you test that every change introduced in that version works smoothly with your app!

Like

Share



Follow

/hellosuman