Supporting Different Platform Versions

While the latest versions of Android often provide great APIs for your app, you should continue to support older versions of Android until more devices get updated. This lesson shows you how to take advantage of the latest APIs while continuing to support older versions as well.

The dashboard for Platform Versions (http://developer.android.com/about/dashboards/index.html) is updated regularly to show the distribution of active devices running each version of Android, based on the number of devices that visit the Google Play Store. Generally, it's a good practice to support about 90% of the active devices, while targeting your app to the latest version.

THIS LESSON TEACHES YOU TO

1. Specify Minimum and Target API Levels
2. Check System Version at Runtime
3. Use Platform Styles and Themes

YOU SHOULD ALSO READ

- Android API Levels
- Android Support Library

> **Tip:** In order to provide the best features and functionality across several Android versions, you should use the Android Support Library (/tools/extras/support-library.html) in your app, which allows you to use several recent platform APIs on older versions.

Specify Minimum and Target API Levels

The AndroidManifest.xml (/guide/topics/manifest/manifest-intro.html) file describes details about your app and identifies which versions of Android it supports. Specifically, the `minSdkVersion` and `targetSdkVersion` attributes for the `<uses-sdk` (/guide/topics/manifest/uses-sdk-element.html) element identify the lowest API level with which your app is compatible and the highest API level against which you've designed and tested your app.

For example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

As new versions of Android are released, some style and behaviors may change. To allow your app to take advantage of these changes and ensure that your app fits the style of each user's device, you should set the `targetSdkVersion` (/guide/topics/manifest/uses-sdk-element.html#target) value to match the latest Android version available.

Check System Version at Runtime

Android provides a unique code for each platform version in the `Build` (/reference/android/os/Build.html) constants class. Use these codes within your app to build conditions that ensure the code that depends on higher API levels is executed only when those APIs are available on the system.

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

> **Note:** When parsing XML resources, Android ignores XML attributes that aren't supported by the current

> device. So you can safely use XML attributes that are only supported by newer versions without worrying about older versions breaking when they encounter that code. For example, if you set the `targetSdkVersion="11"`, your app includes the `ActionBar` (/reference/android/app/ActionBar.html) by default on Android 3.0 and higher. To then add menu items to the action bar, you need to set `android:showAsAction="ifRoom"` in your menu resource XML. It's safe to do this in a cross-version XML file, because the older versions of Android simply ignore the `showAsAction` attribute (that is, you *do not* need a separate version in `res/menu-v11/`).

Use Platform Styles and Themes

Android provides user experience themes that give apps the look and feel of the underlying operating system. These themes can be applied to your app within the manifest file. By using these built in styles and themes, your app will naturally follow the latest look and feel of Android with each new release.

To make your activity look like a dialog box:

```
<activity android:theme="@android:style/Theme.Dialog">
```

To make your activity have a transparent background:

```
<activity android:theme="@android:style/Theme.Translucent">
```

To apply your own custom theme defined in `/res/values/styles.xml`:

```
<activity android:theme="@style/CustomTheme">
```

To apply a theme to your entire app (all activities), add the `android:theme` attribute to the `<application>` (/guide/topics/manifest/application-element.html) element:

```
<application android:theme="@style/CustomTheme">
```

For more about creating and using themes, read the Styles and Themes (/guide/topics/ui/themes.html) guide.