About permissions

• Every Android app runs in a limited-access sandbox
• If an app needs to use resources or information outside of its own sandbox, the app has to request the appropriate permission
• You declare that your app needs a permission by listing the permission in the App Manifest
Add Permissions to the Manifest

• To declare that your app needs a permission, put a `<uses-permission>` element in your app manifest, as a child of the top-level `<manifest>` element.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.snazzyapp">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application ...>
    ...
    </application>
</manifest>
```

A permission for an app that needs to send SMS messages

Requesting permissions at run time

• Since Android 6.0 (API level 23), users grant permissions while the app is running, not upon install.
• Why? “to streamline the app install process, and to gives the user more control over the app's functionality”
• The user can revoke the permissions at any time, by going to the app's Settings screen.
Normal vs. dangerous permissions

• System permissions are divided into two categories, **normal** and **dangerous**
• Normal permissions do not directly risk the user's privacy, and are granted permission **automatically**
• Dangerous permissions can give the app access to the user's confidential data, and should **explicitly approved** by the user

For example, a request for turning on the device's **flashlight**

For example, a request for reading user's **contacts**

Normal vs. dangerous permissions (2)

• **On all versions**, both normal and dangerous permissions should be declared in the manifest
• However, the **effect** of that declaration is different depending on the system version and your app's target SDK level

For a list of system permissions, see **Normal and Dangerous Permissions**
Android 5.1 or lower

- If the device is running Android 5.1 or lower, or your app's target SDK is 22 or lower:
  - If you list a **dangerous** permission in your manifest, the user has to grant the permission when they install the app
  - If they do not grant the permission, the system does not install the app at all

Android 6.0 or higher

- If the device is running Android 6.0 or higher, and your app's target SDK is 23 or higher:
  - The app has to list the permissions in the manifest, and it must request each dangerous permission it needs while the app is running
  - The user can grant or deny each permission, and the app can continue to run with limited capabilities even if the user denies a permission request

**Note:** Beginning with Android 6.0 (API level 23), users can revoke permissions from any app at any time, **even if the app targets a lower API level**. You should test your app to verify that it behaves properly when it's missing a needed permission, regardless of what API level your app targets.

Following, we show how to check for, and request permissions using the **Support Library**, which is the recommended way (uniform handling).
Check for permissions

• To check if you have a permission, call the `ContextCompat.checkSelfPermission()` method
• If your app needs a **dangerous** permission, you must check whether you have that permission **every time** you perform an operation that requires that permission
• The user is always free to revoke the permission, so even if the app used the camera yesterday, it can't assume it still has that permission today

Example: check if the activity has permission to write to the calendar

```java
// Assume thisActivity is the current activity
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity, Manifest.permission.WRITE_CALENDAR);
```

• If the app has the permission, the method returns `PackageManager.PERMISSION_GRANTED`, and the app can proceed with the operation
• Otherwise, the method returns `PERMISSION_DENIED`, and the app has to explicitly ask the user for permission

Android provides several methods you can use to request a permission. Calling these methods brings up a standard **Android dialog**, which you cannot customize.
Explaining the rationale to the user

- Before you request a permission, you should consider providing an explanation to the user.
- To help find situations where the user might need an explanation, Android provides a utility method, `shouldShowRequestPermissionRationale()`.
- This method returns true if the app has requested this permission previously and the user denied the request.

**Note:** If the user turned down the permission request in the past and chose the *Don't ask again* option in the permission request system dialog, this method returns false.

---

**Step 1: request the permission you need**

- Do we have a permission to read from Contacts?
- Should we show an explanation to the user?
- In case no explanation is needed, we request the permission.

```java
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.READ_CONTACTS) != PackageManager.PERMISSION_GRANTED) {

    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
        Manifest.permission.READ_CONTACTS)) {
        // Show an explanation to the user "asynchronously" -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.
    } else {
        // No explanation needed, we can request the permission.

        ActivityCompat.requestPermissions(thisActivity,
            new String[]{Manifest.permission.READ_CONTACTS},
            MY_PERMISSIONS_REQUEST_READ_CONTACTS);

        // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
        // app-defined int constant. The callback method gets the
        // result of the request.
    }
}
```
Step 2: handle the request response (callback)

override this method to find out whether the permission was granted

The callback is passed the same request code you passed to requestPermissions()

See this link for permissions’ best practices to prevent bad user experience.

```java
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0 
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // permission was granted, yay! Do the 
                // contacts-related task you need to do.
                
            } else {
                // permission denied, boo! Disable the 
                // functionality that depends on this permission.
                
            }
    }

    // other 'case' lines to check for other 
    // permissions this app might request
}
```