Requesting Permissions at Run Time

This lesson teaches you to

- Check For Permissions
- Request Permissions

Dependencies and Prerequisites

- Android 6.0 (API level 23)

You should also read

- Normal and Dangerous Permissions

Beginning in Android 6.0 (API level 23), users grant permissions to apps while the app is running, not when they install the app. This approach streamlines the app install process, since the user does not need to grant permissions when they install or update the app. It also gives the user more control over the app's functionality; for example, a user could choose to give a camera app access to the camera but not to the device location. The user can revoke the permissions at any time, by going to the app's Settings screen.

System permissions are divided into two categories, *normal* and *dangerous:*

- Normal permissions do not directly risk the user's privacy. If your app lists a normal permission in its manifest, the system grants the permission automatically.
- Dangerous permissions can give the app access to the user's confidential data. If your app lists a normal permission in its manifest, the system grants the permission automatically. If you list a dangerous permission, the user has to explicitly give approval to your app.

For more information, see Normal and Dangerous Permissions (https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous).

the *effect* of that declaration is different depending on the system version and your app's target SDK level:

- 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.
- 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.

This lesson describes how to use the Android Support Library (https://developer.android.com/tools/support-library/index.html) to check for, and request, permissions. The Android framework provides similar methods as of Android 6.0 (API level 23). However, using the support library is simpler, since your app doesn't need to check which version of Android it's running on before calling the methods.

Check For Permissions

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.

To check if you have a permission, call the `ContextCompat.checkSelfPermission()` (https://developer.android.com/reference/android/support/v4/content/ContextCompat.html#checkSelfPermission(android.cont`Context, java.lang.String))` method. For example, this snippet shows how to check if the activity has permission to write to the calendar:

```
// 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` (https://developer.android.com/reference/android/content/pm/PackageManager.html#PERMISSION_GRANTED), and the app can proceed with the operation. If the app does not have the permission, the method returns `PERMISSION_DENIED` (https://developer.android.com/reference/android/content/pm/PackageManager.html#PERMISSION_DENIED), and the app has to

Request Permissions

If your app needs a dangerous permission that was listed in the app manifest, it must ask the user to grant the 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.

Explain why the app needs permissions

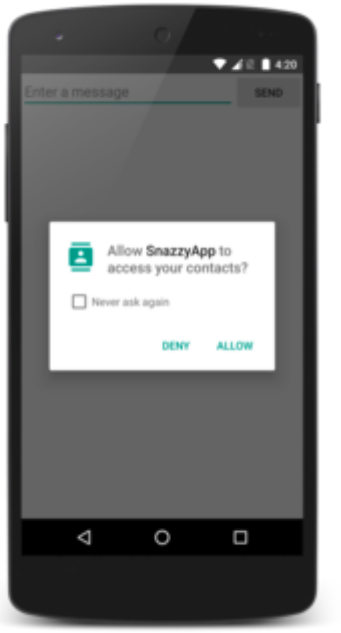

Figure 1. System dialog prompting the user to grant or deny a permission.

In some circumstances, you might want to help the user understand why your app needs a permission. For example, if a user launches a photography app, the user probably won't be surprised that the app asks for permission to use the camera, but the user might not understand why the app wants access to the user's location or contacts. Before you request a permission, you should consider providing an explanation to the user. Keep in mind that you don't want to overwhelm the user with explanations; if you provide too many explanations, the user might find the app frustrating and

One approach you might use is to provide an explanation only if the user has already turned down that permission request. If a user keeps trying to use functionality that requires a permission, but keeps turning down the permission request, that probably shows that the user doesn't understand why the app needs the permission to provide that functionality. In a situation like that, it's probably a good idea to show an explanation.

To help find situations where the user might need an explanation, Android provides a utiltity method, `shouldShowRequestPermissionRationale()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#shouldShowRequestPermissionRationaldroid.app.Activity, java.lang.String).) 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`. The method also returns `false` if a device policy prohibits the app from having that permission.

Request the permissions you need

If your app doesn't already have the permission it needs, the app must call one of the `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Acti, java.lang.String[], int)) methods to request the appropriate permissions. Your app passes the permissions it wants, and also an integer *request code* that you specify to identify this permission request. This method functions asynchronously: it returns right away, and after the user responds to the dialog box, the system calls the app's callback method with the results, passing the same request code that the app passed to `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Acti, java.lang.String[], int)).

The following code checks if the app has permission to read the user's contacts, and requests the permission if necessary:

```
// 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 {
```

```
            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.
    }
}
```

> **Note:** When your app calls `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Activity, java.lang.String[], int)), the system shows a standard dialog box to the user. Your app *cannot* configure or alter that dialog box. If you need to provide any information or explanation to the user, you should do that before you call `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Activity, java.lang.String[], int)), as described in Explain why the app needs permissions (#explain).

Handle the permissions request response

When your app requests permissions, the system presents a dialog box to the user. When the user responds, the system invokes your app's `onRequestPermissionsResult()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.OnRequestPermissionsResultCallback.html#onRequestPermissionsResult(int, java.lang.String[], int[])) method, passing it the user response. Your app has to override that method to find out whether the permission was granted. The callback is passed the same request code you passed to `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Activity, java.lang.String[], int)). For example, if an app requests `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) access it might have the following callback method:

```
@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.
```

```
            }
            return;
        }

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

The dialog box shown by the system describes the permission group (https://developer.android.com/guide/topics/security/permissions.html#perm-groups) your app needs access to; it does not list the specific permission. For example, if you request the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission, the system dialog box just says your app needs access to the device's contacts. The user only needs to grant permission once for each permission group. If your app requests any other permissions in that group (that are listed in your app manifest), the system automatically grants them. When you request the permission, the system calls your `onRequestPermissionsResult()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.OnRequestPermissionsResultCallback.html#onRequestPermissionsResult(int, java.lang.String[], int[])) callback method and passes `PERMISSION_GRANTED` (https://developer.android.com/reference/android/content/pm/PackageManager.html#PERMISSION_GRANTED), the same way it would if the user had explicitly granted your request through the system dialog box.

> **Note:** Your app still needs to explicitly request every permission it needs, even if the user has already granted another permission in the same group. In addition, the grouping of permissions into groups may change in future Android releases. Your code should not rely on the assumption that particular permissions are or are not in the same group.

For example, suppose you list both `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) and `WRITE_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS) in your app manifest. If you request `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) and the user grants the permission, and you then request `WRITE_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS), the system immediately grants you that permission without interacting with the user.

If the user denies a permission request, your app should take appropriate action. For example, your app might show a dialog explaining why it could not perform the user's requested action that needs that permission.

When the system asks the user to grant a permission, the user has the option of telling the system not to ask for that permission again. In that case, any time an app uses `requestPermissions()` (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Acti

questPermissionsResult(int, java.lang.String[], int[])callback method and passes PERMISSION_DENIED (https://developer.android.com/reference/android/content/pm/PackageManager.html#PERMISSION_DENIED) the same way it would if the user had explicitly rejected your request again. This means that when you call requestPermissions() (https://developer.android.com/reference/android/support/v4/app/ActivityCompat.html#requestPermissions(android.app.Acti , java.lang.String[], int)),you cannot assume that any direct interaction with the user has taken place.