66. Making Runtime Permission Requests in Android

In a number of the example projects created in preceding chapters, changes have been made to the AndroidManifest.xml file to request permission for the app to perform a specific task. In a couple of instances, for example, internet access permission has been requested in order to allow the app to download and display web pages. In each case up until this point, the addition of the request to the manifest was all that is required in order for the app to obtain permission from the user to perform the designated task.

There are, however, a number of permissions for which additional steps are required in order for the app to function when running on Android 6.0 or later. The first of these so-called "dangerous" permissions will be encountered in the next chapter. Before reaching that point, however, this chapter will outline the steps involved in requesting such permissions when running on the latest generations of Android.

66.1 Understanding Normal and Dangerous Permissions

Android enforces security by requiring the user to grant permission for an app to perform certain tasks. Prior to the introduction of Android 6, permission was always sought at the point that the app was installed on the device. Figure 66-1, for example, shows a typical screen seeking a variety of permissions during the installation of an app via Google Play.

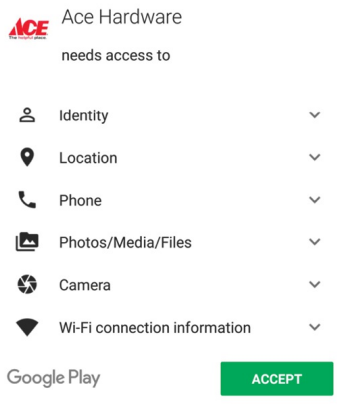

Figure 66-1

For many types of permissions this scenario still applies for apps on Android 6.0 or later. These permissions are referred to as *normal permissions* and are still required to be accepted by the user at the point of installation. A second type of permission, referred to as *dangerous permissions* must also be declared within the manifest file in the same way as a normal permission, but must also be requested from the user when the application is first launched. When such a request is made, it appears in the form of a dialog box as illustrated in Figure 66-2:

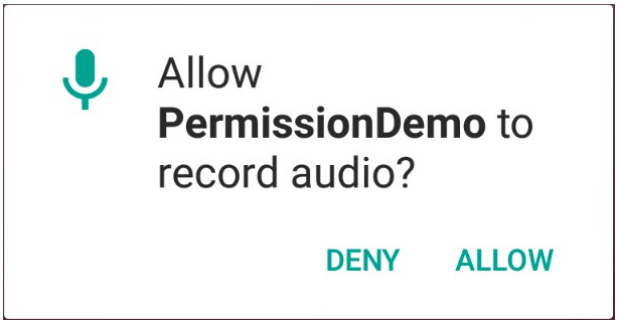

Figure 66-2

The full list of permissions that fall into the dangerous category is contained in Table 66-3:

Permission Group	Permission
Calendar	READ_CALENDAR WRITE_CALENDAR
Camera	CAMERA
Contacts	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
Location	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
Microphone	RECORD_AUDIO
Phone	READ_PHONE_STATE CALL_PHONE READ_CALL_LOG

	WRITE_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS
Sensors	BODY_SENSORS
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
Storage	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Table 66-3

66.2 Creating the Permissions Example Project

Create a new project in Android Studio, entering *PermissionDemo* into the Application name field and *com.ebookfrenzy* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 19: Android 4.4 (KitKat). Continue to proceed through the screens, requesting the creation of an Empty Activity named *PermissionDemoActivity* with a corresponding layout named *activity_permission_demo.*

66.3 Checking for a Permission

The Android Support Library contains a number of methods that can be used to seek and manage dangerous permissions within the code of an Android app. These API calls can be made safely regardless of the version of Android on which the app is running, but will only perform meaningful tasks when executed on Android 6.0 or later.

Before an app attempts to make use of a feature that requires approval of a dangerous permission, and regardless of whether or not permission was previously granted, the code must check that the permission has been

granted. This can be achieved via a call to the *checkSelfPermission()* method of the ContextCompat class, passing through as arguments a reference to the current activity and the permission being requested. The method will check whether the permission has been previously granted and return an integer value matching *PackageManager.PERMISSION_GRANTED* or *PackageManager.PERMISSION_DENIED*.

Within the *PermissionDemoActivity.java* file of the example project, modify the code to check whether permission has been granted for the app to record audio:

```
package com.ebookfrenzy.permissiondemoactivity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.Manifest;
import android.content.pm.PackageManager;
import android.support.v4.content.ContextCompat;
import android.util.Log;

public class PermissionDemoActivity extends AppCompatActivity {

    private static String TAG = "PermissionDemo";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_permission_demo);

        setupPermissions();
    }

    private void setupPermissions() {
        int permission = ContextCompat.checkSelfPermission(this,
                Manifest.permission.RECORD_AUDIO);

        if (permission != PackageManager.PERMISSION_GRANTED) {
            Log.i(TAG, "Permission to record denied");
        }
    }
}
```

Run the app on a device or emulator running a version of Android that

predates Android 6.0 and check the log cat output within Android Studio. After the app has launched, the Logcat output should include the "Permission to record denied" message.

Edit the *AndroidManifest.xml* file (located in the Project tool window under *app -> manifests*) and add a line to request recording permission as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ebookfrenzy.permissiondemoactivity" >

    <uses-
permission android:name="android.permission.RECORD_AUDIO" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".PermissionDemoActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
                  android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Compile and run the app once again and note that this time the permission denial message does not appear. Clearly, everything that needs to be done to request this permission on older versions of Android has been done. Run the app on a device or emulator running Android 6.0 or later, however, and note that even though permission has been added to the manifest file, the check still reports that permission has been denied. This is because Android version 6 or later requires that the app also request dangerous permissions at runtime.

66.4 Requesting Permission at Runtime

A permission request is made via a call to the *requestPermissions()* method of the ActivityCompat class. When this method is called, the permission request is handled asynchronously and a method named *onRequestPermissionsResult()* is called when the task is completed.

The *requestPermissions()* method takes as arguments a reference to the current activity, together with the identifier of the permission being requested and a request code. The request code can be any integer value and will be used to identify which request has triggered the call to the *onRequestPermissionsResult()* method. Modify the *PermissionDemoActivity.java* file to declare a request code and request recording permission in the event that the permission check failed:

```
package com.ebookfrenzy.permissiondemoactivity;

import android.Manifest;
import android.content.pm.PackageManager;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.support.v4.app.ActivityCompat;

public class PermissionDemoActivity extends AppCompatActivity {

    private static String TAG = "PermissionDemo";
    private static final int RECORD_REQUEST_CODE = 101;
.
.
    @Override
    private void setupPermissions() {

        int permission = ContextCompat.checkSelfPermission(this,
                Manifest.permission.RECORD_AUDIO);

        if (permission != PackageManager.PERMISSION_GRANTED) {
            Log.i(TAG, "Permission to record denied");
            makeRequest();
        }
    }

    protected void makeRequest() {
```

```
        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.RECORD_AUDIO},
                RECORD_REQUEST_CODE);
    }
}
```

Next, implement the *onRequestPermissionsResult()* method so that it reads as follows:

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[]
grantResults) {
    switch (requestCode) {
        case RECORD_REQUEST_CODE: {

            if (grantResults.length == 0
                    || grantResults[0] !=
                     PackageManager.PERMISSION_GRANTED) {

                Log.i(TAG, "Permission has been denied by user");
            } else {
                Log.i(TAG, "Permission has been granted by user");
            }
        }
    }
}
```

Compile and run the app on an Android 6 or later emulator or device and note that a dialog seeking permission to record audio appears as shown in Figure 66-3:

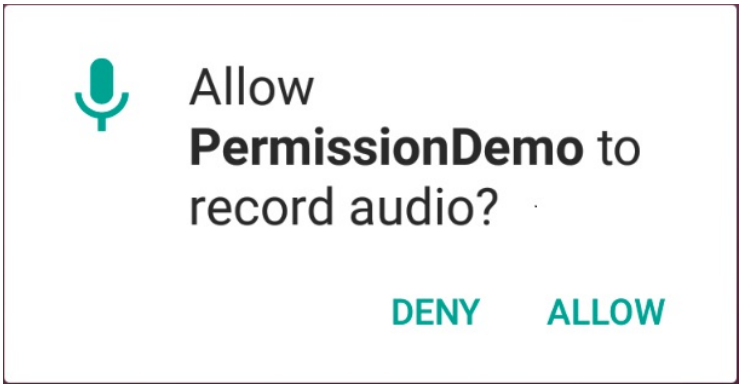

Figure 66-3

Tap the Allow button and check that the “Permission has been granted by user” message appears in the Logcat panel.

Once the user has granted the requested permission, the *checkSelfPermission()* method call will return a PERMISSION_GRANTED result on future app invocations until the user uninstalls and re-installs the app or changes the permissions for the app in Settings.

66.5 Providing a Rationale for the Permission Request

As is evident from Figure 66-3, the user has the option to deny the requested permission. In this case, the app will continue to request the permission each time that it is launched by the user unless the user selected the "Never ask again" option prior to clicking on the Deny button. Repeated denials by the user may indicate that the user doesn't understand why the permission is required by the app. The user might, therefore, be more likely to grant permission if the reason for the requirements is explained when the request is made. Unfortunately, it is not possible to change the content of the request dialog to include such an explanation.

An explanation is best included in a separate dialog which can be displayed before the request dialog is presented to the user. This raises the question as to when to display this explanation dialog. The Android documentation recommends that an explanation dialog only be shown in the event that the user has previously denied the permission and provides a method to identify when this is the case.

A call to the *shouldShowRequestPermissionRationale()* method of the ActivityCompat class will return a true result if the user has previously denied a request for the specified permission, and a false result if the request has not previously been made. In the case of a true result, the app should display a dialog containing a rationale for needing the permission and, once the dialog has been read and dismissed by the user, the permission request should be repeated.

To add this functionality to the example app, modify the *onCreate()* method so that it reads as follows:

```
.
.
import android.app.AlertDialog;
import android.content.DialogInterface;
.
.
```

```
private void setupPermissions() {

    int permission = ContextCompat.checkSelfPermission(this,
            Manifest.permission.RECORD_AUDIO);

    if (permission != PackageManager.PERMISSION_GRANTED) {
        Log.i(TAG, "Permission to record denied");

        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.RECORD_AUDIO)) {
            AlertDialog.Builder builder =
                    new AlertDialog.Builder(this);
            builder.setMessage("Permission to access the microphone
is required for this app to record audio.")
                    .setTitle("Permission required");

            builder.setPositiveButton("OK",
                    new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int id) {
                    Log.i(TAG, "Clicked");
                    makeRequest();
                }
            });

            AlertDialog dialog = builder.create();
            dialog.show();
        } else {
            makeRequest();
        }
    }
}
```

The method still checks whether or not the permission has been granted, but now also identifies whether a rationale needs to be displayed. If the user has previously denied the request, a dialog is displayed containing an explanation and an OK button on which a listener is configured to call the *makeRequest()* method when the button is tapped. In the event that the permission request has not previously been made, the code moves directly to seeking permission.

66.6 Testing the Permissions App

On the Android 6 or later device or emulator session on which testing is

being performed, launch the Settings app, select the Apps option and scroll to and select the PermissionDemo app. On the app settings screen, tap the uninstall button to remove the app from the device.

Run the app once again and, when the permission request dialog appears, click on the Deny button. Terminate the app, run it a second time and verify that the rationale dialog appears. Tap the OK button and, when the permission request dialog appears, tap the Allow button.

Return to the Settings app, select the Apps option and select the PermissionDemo app once again from the list. Once the settings for the app are listed, verify that the Permissions section lists the *Microphone* permission:

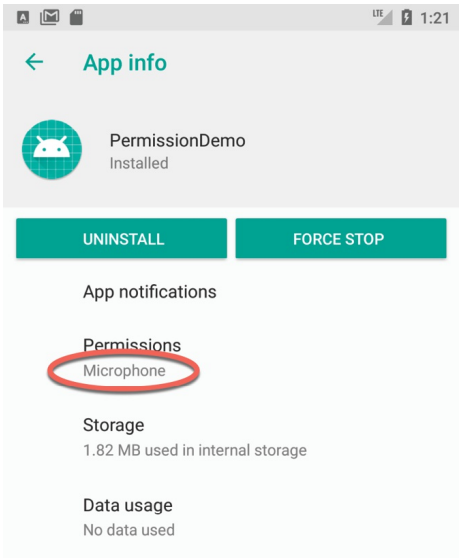

Figure 66-4

66.7 Summary

Prior to the introduction of Android 6.0 the only step necessary for an app to request permission to access certain functionality was to add an appropriate line to the application's manifest file. The user would then be prompted to approve the permission at the point that the app was installed. This is still the case for most permissions, with the exception of a set of permissions that are considered dangerous. Permissions that are considered dangerous usually have the potential to allow an app to violate the user's privacy such as allowing access to the microphone, contacts list or external storage.

As outlined in this chapter, apps based on Android 6 or later must now request dangerous permission approval from the user when the app launches

in addition to including the permission request in the manifest file.