Android Development

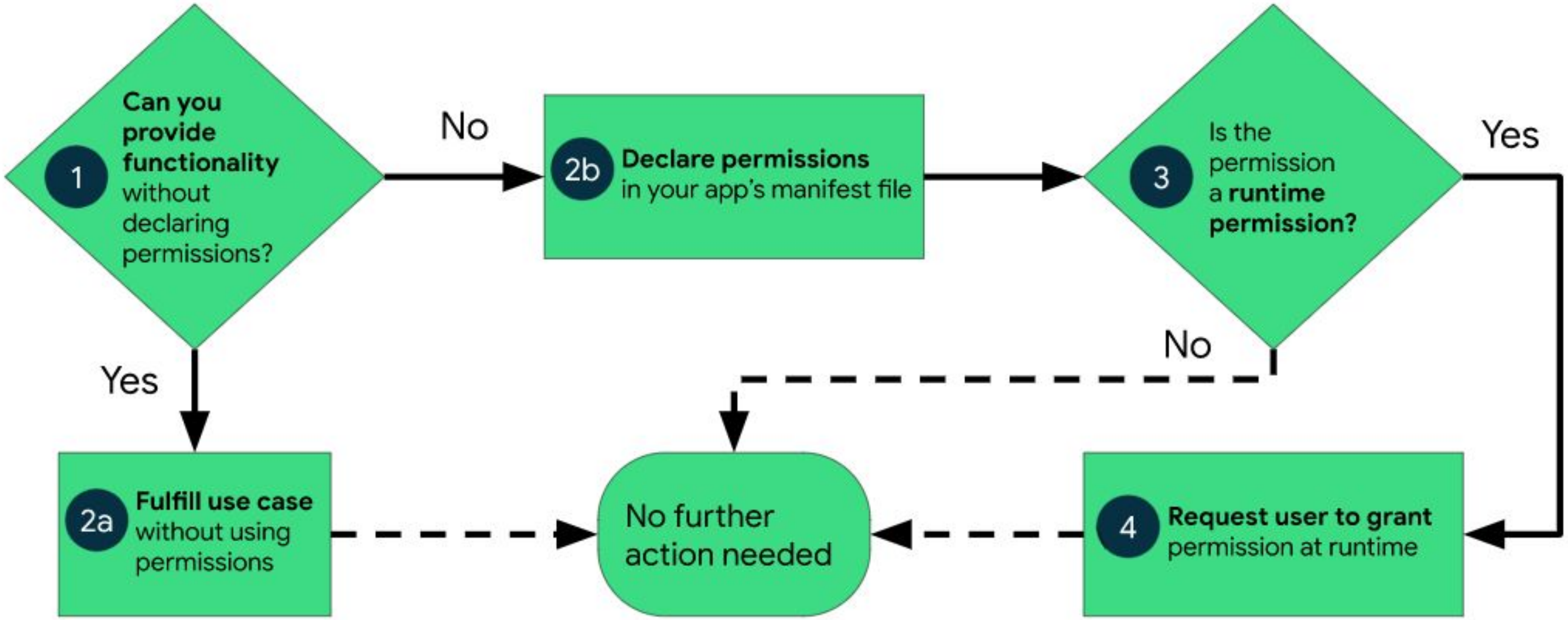
Android Permissions

Android protects the access of

- **Restricted data**, such as system state and a user's contact information.
- **Restricted actions**, such as connecting to a paired device and recording audio.

Steps

- **Declare Permissions**
- **Request Runtime**
- **Handle Permissions Authorization**


1
Can you provide functionality without declaring permissions?
No
2b
Declare permissions in your app's manifest file
3
Is the permission a runtime permission?
Yes
Yes
No
2a
Fulfill use case without using permissions
No further action needed
4
Request user to grant permission at runtime

Android Permissions

- Install time permissions
- Normal permissions
- Signature permissions

Normal permissions

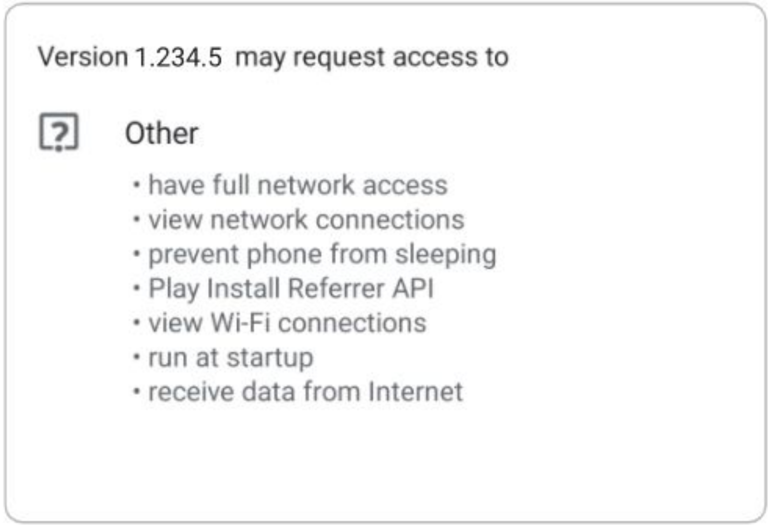
- Runtime permissions
- Special permissions

Runtime permissions

Android Permissions

Install Time

When you declare install-time permissions in your app, the system automatically grants your app the permissions when the user installs your app.



Version 1.234.5 may request access to

Other

- have full network access
- view network connections
- prevent phone from sleeping
- Play Install Referrer API
- view Wi-Fi connections
- run at startup
- receive data from Internet

Android Permissions

Normal Permissions

These permissions allow access to data and actions that extend beyond your app's sandbox. However, the data and actions present very little risk to the user's privacy, and the operation of other apps.

Normal permissions include:
ACCESS_NOTIFICATION_POLICY (?),
ACCESS_WIFI_STATE, BLUETOOTH,
INTERNET,
KILL_BACKGROUND_PROCESSES,
MANAGE_OWN_CALLS,
MODIFY_AUDIO_SETTINGS,
SET_ALARM,
SET_WALLPAPER,
VIBRATE etc.

https://developer.android.com/reference/android/Manifest.permission

Android Permissions - Normal Permissions

Normal Permissions include the following permissions:

1. ACCESS_LOCATION_EXTRA_COMMANDS
2. ACCESS_NETWORK_STATE
3. CHANGE_NETWORK_STATE
4. ACCESS_WIFI_STATE
5. CHANGE_WIFI_STATE
6. CHANGE_WIFI_MULTICAST_STATE
7. BLUETOOTH
8. BLUETOOTH_ADMIN
9. INTERNET
10. SET_ALARM
11. SET_WALLPAPER
12. VIBRATE
13. WAKE_LOCK

Android Permissions

Signature Permissions

Signature protection level permissions are automatically granted to any requesting app signed with the same key (as the app that defines the permission). This improves the user experience, as the user does not have to explicitly grant the permission to a requesting app, whilst at the same time it prevents other apps (not by the same developer) from acquiring the permission.

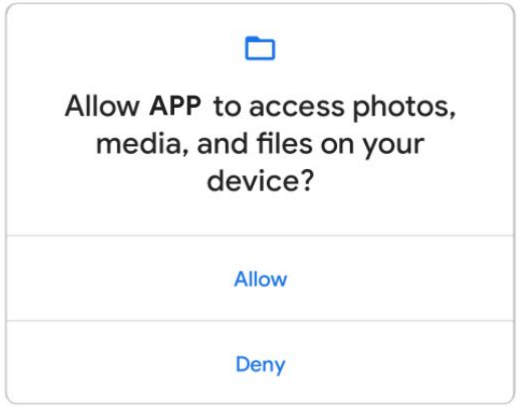
Some of the Signature permissions are as follows:

1. BIND_ACCESSIBILITY_SERVICE
2. BIND_AUTOFILL_SERVICE
3. BIND_CARRIER_SERVICE
4. BIND_DEVICE_ADMIN
5. BIND_INPUT_METHOD
6. BIND_NFC_SERVICE
7. BIND_TV_INPUT
8. BIND_WALLPAPER
9. READ_VOICEMAIL
10. WRITE_SETTINGS
11. WRITE_VOICEMAIL

Android Permissions

Runtime Permissions

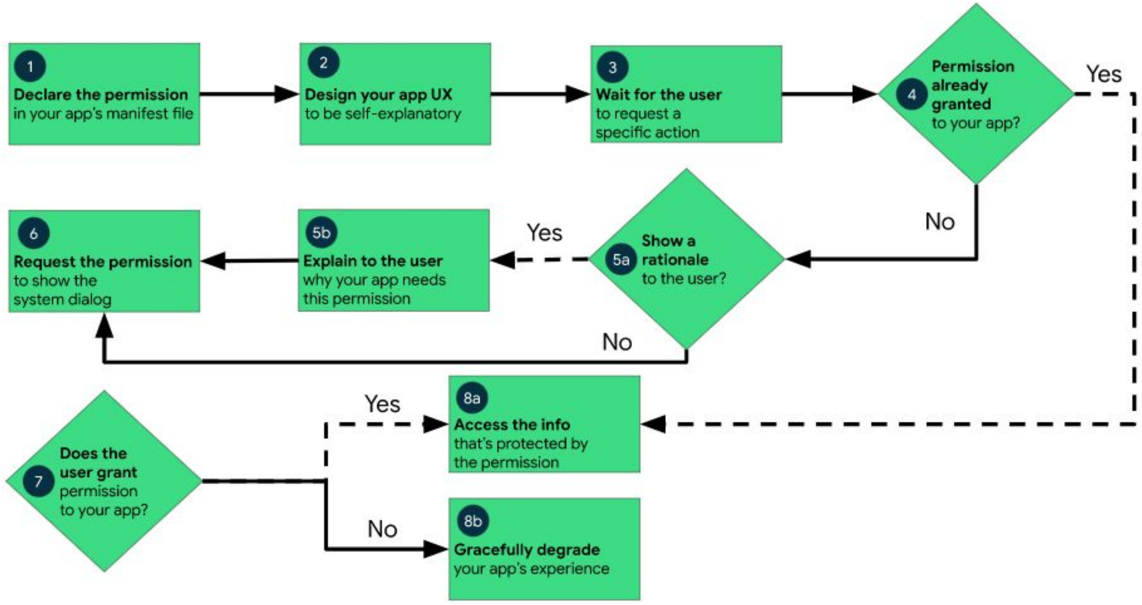
Runtime permissions, also known as dangerous permissions, give your app additional access to restricted data, and they allow your app to perform restricted actions that more substantially affect the system and other apps.


Allow **APP** to access photos, media, and files on your device?

READ_CALENDAR,
WRITE_CALENDAR,
CAMERA,
READ_CONTACTS,
WRITE_CONTACTS,
RECORD_AUDIO,
READ_PHONE_NUMBERS,
CALL_PHONE,
ANSWER_PHONE_CALLS, SEND_SMS, RECEIVE_SMS,
READ_SMS and so on.

Android Permissions

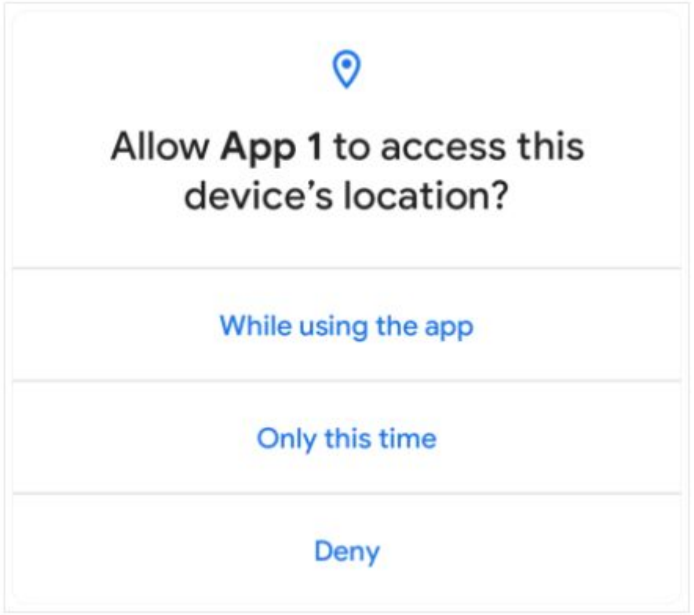
Runtime Permissions


1
Declare the permission in your app's manifest file
2
Design your app UX to be self-explanatory
3
Wait for the user to request a specific action
4
Permission already granted to your app?
Yes
No
5a
Show a rationale to the user?
Yes
5b
Explain to the user why your app needs this permission
6
Request the permission to show the system dialog
No
7
Does the user grant permission to your app?
Yes
8a
Access the info that's protected by the permission
No
8b
Gracefully degrade your app's experience

Android Permissions

One-time permissions

Starting in Android 11 (API level 30), whenever your app requests a permission related to location, microphone, or camera, the user-facing permissions dialog contains an option called **Only this time**, If the user selects this option in the dialog, your app is granted a temporary *one-time permission*



Android Permissions

```
val requestPermissionLauncher =
    registerForActivityResult(RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            // Permission is granted. Continue the action or workflow in your
            // app.
        } else {
            // Explain to the user that the feature is unavailable because the
            // features requires a permission that the user has denied. At the
            // same time, respect the user's decision. Don't link to system
            // settings in an effort to convince the user to change their
            // decision.
        }
    }
```

Android Permissions

```kotlin
when {
    ContextCompat.checkSelfPermission(
            CONTEXT,
            Manifest.permission.REQUESTED_PERMISSION
            ) == PackageManager.PERMISSION_GRANTED -> {
        // You can use the API that requires the permission.
    }
    shouldShowRequestPermissionRationale(...) -> {
        // In an educational UI, explain to the user why your app requires this
        // permission for a specific feature to behave as expected. In this UI,
        // include a "cancel" or "no thanks" button that allows the user to
        // continue using your app without granting the permission.
        showInContextUI(...)
    }
    else -> {
        // You can directly ask for the permission.
        // The registered ActivityResultCallback gets the result of this request.
        requestPermissionLauncher.launch(
                Manifest.permission.REQUESTED_PERMISSION)
    }
}
```

Android Permissions - RequestCode

```
when {
    ContextCompat.checkSelfPermission(
            CONTEXT,
            Manifest.permission.REQUESTED_PERMISSION
            ) == PackageManager.PERMISSION_GRANTED -> {
        // You can use the API that requires the permission.
        performAction(...)
    }
    shouldShowRequestPermissionRationale(...) -> {
        // In an educational UI, explain to the user why your app requires this
        // permission for a specific feature to behave as expected. In this UI,
        // include a "cancel" or "no thanks" button that allows the user to
        // continue using your app without granting the permission.
        showInContextUI(...)
    }
    else -> {
        // You can directly ask for the permission.
        requestPermissions(CONTEXT,
                arrayOf(Manifest.permission.REQUESTED_PERMISSION),
                REQUEST_CODE)
    }
}
```

Android Permissions - RequestCode

```
override fun onRequestPermissionsResult(requestCode: Int,
        permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        PERMISSION_REQUEST_CODE -> {
            // If request is cancelled, the result arrays are empty.
            if ((grantResults.isNotEmpty() &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED)) {
                // Permission is granted. Continue the action or workflow
                // in your app.
            } else {
                // Explain to the user that the feature is unavailable because
                // the features requires a permission that the user has denied.
                // At the same time, respect the user's decision. Don't link to
                // system settings in an effort to convince the user to change
                // their decision.
            }
            return
        }

        // Add other 'when' lines to check for other
        // permissions this app might request.
        else -> {
            // Ignore all other requests.
        }
    }
}
```

Tips

Android auto-resets permissions of unused apps

If your app targets Android 11 (API level 30) or higher and isn't used for a few months, the system protects user data by automatically resetting the sensitive runtime permissions that the user had granted your app. Learn more in the guide about app hibernation.

Determine whether your app was already granted the permission

To check if the user has already granted your app a particular permission, pass that permission into the ContextCompat.checkSelfPermission() method. This method returns either PERMISSION_GRANTED or PERMISSION_DENIED, depending on whether your app has the permission.