Restrictions on starting activities from the background

Archived: 2026-04-05 21:44:47 UTC

Android 10 (API level 29) and higher place restrictions on when apps can start activities when the app runs in the background. These restrictions help minimize interruptions for the user and keep the user more in control of what's shown on their screen.

This guide presents notifications as an alternative for starting activities from the background. It also lists the specific cases where the restriction doesn't apply.

Display notifications instead

In nearly all cases, apps in the background must display time-sensitive notifications to provide urgent information to the user instead of directly starting an activity. Such notifications include handling an incoming phone call or an active alarm clock.

This notification-based alert and reminder system provides several advantages for users:

- When using the device, the user sees a heads-up notification that lets them respond. The user maintains their current context and has control over the content that they see on the screen.
- Time-sensitive notifications respect the user's Do Not Disturb rules. For example, users might permit calls only from specific contacts or from repeat callers when Do Not Disturb is enabled.
- When the device's screen is off, the full-screen intent launches immediately.
- In the device's **Settings** screen, the user can see which apps have recently sent notifications, including from specific notification channels. From this screen, the user can control their notification preferences.

When apps can start activities

Apps running on Android 10 or higher can start activities when one or more of the following conditions are met:

- The app has a visible window, such as an activity in the foreground.
- The app has an activity in the back stack of the foreground task.
- The app has an activity in the back stack of an existing task on the Recents screen.
- The app has an activity that started very recently.
- The app called `finish()` on an activity very recently. This applies only when the app had either an activity in the foreground or an activity in the back stack of the foreground task at the time `finish()` was called.
- The app has one of the following services that is bound by the system. These services might need to launch a UI.

    - `AccessibilityService`
    - `AutofillService`
    - `CallRedirectionService`
    - `HostApduService`
    - `InCallService`
    - `TileService` (Not applicable in Android 14 (API level 34) and higher)
    - `VoiceInteractionService`
    - `VrListenerService`.

- The app has a service that is bound by a different, visible app. The app bound to the service must remain visible for the app in the background to start activities successfully.
- The app receives a notification `PendingIntent` from the system. In the case of pending intents for services and broadcast receivers, the app can start activities for a few seconds after the pending intent is sent.
- The app receives a `PendingIntent` that is sent from a different, visible app.
- The app receives a system broadcast where the app is expected to launch a UI. Examples include `ACTION_NEW_OUTGOING_CALL` and `SECRET_CODE_ACTION`. The app can start activities for a few seconds after the broadcast is sent.
- The app is associated with a companion hardware device through the `CompanionDeviceManager` API. This API lets the app start activities in response to actions that the user performs on a paired device.
- The app is a device policy controller running in device owner mode. Example use cases include fully managed enterprise devices as well as dedicated devices like digital signage and kiosks.
- The app is granted the `SYSTEM_ALERT_WINDOW` permission by the user.

Opt-In required when starting activities from PendingIntents

To avoid allowing accidental Activity starts based on the listed conditions, starting with Android 14 there are explicit APIs that allow you to opt in or out of granting an app permissions for Activity starts.

Apps targeting Android 15 or higher will default to no longer implicitly granting background activity launch (BAL) privileges to `PendingIntents` they create. Explicit opt-in is required, in order to do so, these are the options depending if the app is sending or creating `PendingIntents`.

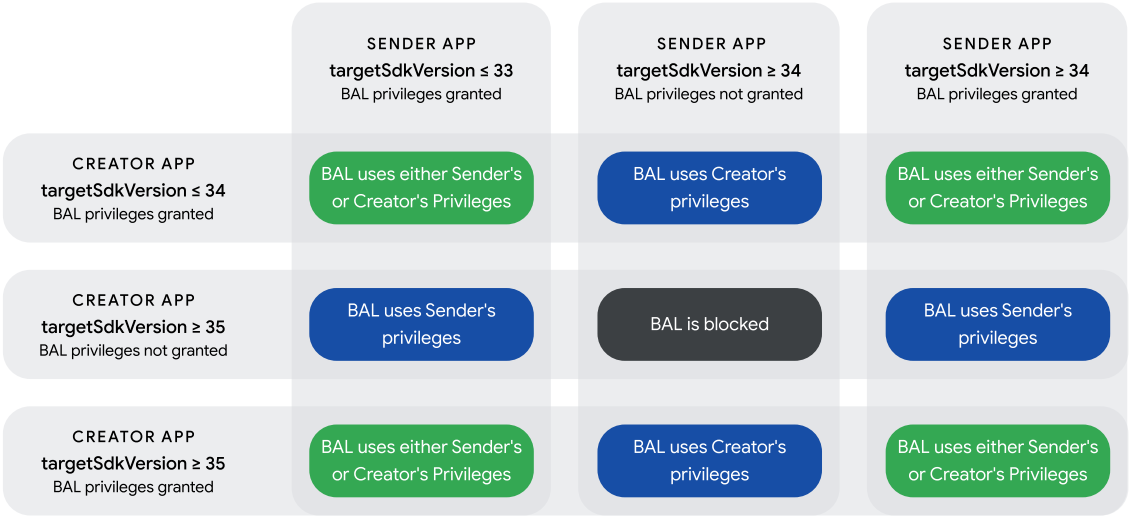

Figure 1: Decision flow for background activity launches.

By the Sender of the PendingIntent

Apps targeting Android 14 or higher that want to start a `PendingIntent` must

- fulfill the listed conditions **and**
- opt in to allow background activity launch based on those exceptions

This opt-in should **only** happen if the app developer knows that the app is going to start an Activity.

To opt in, the app should pass an `ActivityOptions` bundle with `setPendingIntentBackgroundActivityStartMode(ActivityOptions.MODE_BACKGROUND_ACTIVITY_START_ALLOWED)` to the `PendingIntent.send()` or similar methods.

By the Creator of the PendingIntent

Apps targeting Android 15 or higher that create a `PendingIntent` must now **explicitly** opt in to allow background activity launch if they want those `PendingIntents` to be startable under the listed conditions.

In most cases, the app starting the `PendingIntent` should be the one to opt in. However, if the creating app needs to grant these privileges:

- The `PendingIntent` can be started at any time the creating app is visible.
- The `PendingIntent` can be started at any time if the creating app has special privileges.

To opt in, the app should pass an `ActivityOptions` bundle with `setPendingIntentCreatorBackgroundActivityStartMode(ActivityOptions.MODE_BACKGROUND_ACTIVITY_START_ALLOWED)` to the `PendingIntent.getActivity()` or similar methods.

Read the relevant reference documentation for further details:

- `ActivityOptions.setPendingIntentBackgroundActivityStartMode`

- `ActivityOptions.setPendingIntentCreatorBackgroundActivityStartMode`

Strict Mode

Starting with Android 16, the app developer can enable Strict mode to get notified when an activity launch is blocked (or at risk of getting blocked when the app's target SDK is raised).

Example code to enable from early in your Application, Activity, or other application component's `Application.onCreate()` method:

```
 override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    StrictMode.setVmPolicy(
        StrictMode.VmPolicy.Builder()
        .detectBlockedBackgroundActivityLaunch()
        .penaltyLog()
        .build());
    )
}
```

Read the Strict mode documentation for more details.
