# Foreground service types




Beginning with Android 14 (API level 34), you must declare an appropriate service type for each foreground service. That means you must declare the service type in your app manifest, and also request the appropriate foreground service permission for that type (in addition to requesting the `FOREGROUND_SERVICE` permission). Furthermore, depending on the foreground service type, you might have to request runtime permissions before you launch the service.

**Note:** In many cases, there are purpose-built APIs you can use instead of creating a particular type of foreground service. When purpose-built APIs are available, they are usually a better choice than creating a foreground service. These APIs are listed in **Alternatives** sections within the appropriate foreground service types.

## Camera

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`camera`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_CAMERA`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_CAMERA`

**Runtime prerequisites**
Request and be granted the `CAMERA` runtime permission

**Note:** The `CAMERA` runtime permission is subject to while-in-use restrictions. For this reason, you cannot create a `camera` foreground service while your app is in the background, with a few exceptions. For more information, see Restrictions on starting foreground services that need while-in-use permissions.

**Description**
Continue to access the camera from the background, such as video chat apps that allow for multitasking.

## Connected device

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`connectedDevice`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_CONNECTED_DEVICE`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_CONNECTED_DEVICE`

**Runtime prerequisites**
At least one of the following conditions must be true:

- Declare at least one of the following permissions in your manifest:
  - `CHANGE_NETWORK_STATE`
  - `CHANGE_WIFI_STATE`
  - `CHANGE_WIFI_MULTICAST_STATE`
  - `NFC`
  - `TRANSMIT_IR`
- Request and be granted at least one of the following runtime permissions:
  - `BLUETOOTH_CONNECT`
  - `BLUETOOTH_ADVERTISE`
  - `BLUETOOTH_SCAN`
  - `UWB_RANGING`
- Call `UsbManager.requestPermission()`

**Description**
Interactions with external devices that require a Bluetooth, NFC, IR, USB, or network connection.

**Note:** If your app performs a projection or remote messaging operation, use the corresponding media projection or remote messaging type instead.

**Alternatives**
If your app needs to do continuous data transfer to an external device, consider using the companion device manager instead. Use the companion device presence API to help your app stay running while the companion device is in range.

If your app needs to scan for bluetooth devices, consider using the Bluetooth scan API instead.

## Data sync

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`dataSync`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_DATA_SYNC`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_DATA_SYNC`

**Runtime prerequisites**
None

**Description**
Data transfer operations, such as the following:

- Data upload or download
- Backup-and-restore operations
- Import or export operations
- Fetch data
- Local file processing
- Transfer data between a device and the cloud over a network

**Alternatives**
Create user-initiated data transfer jobs to let users start long-running data upload or download tasks.

Use the download manager API to download data from a URI.

Use `BackupManager` to back up or restore data.

For other use-cases, consider WorkManager.

**Note:** In a future version of Android, this foreground service type will be deprecated. We recommend you migrate to one of the listed alternatives.

## Health

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`health`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_HEALTH`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_HEALTH`

**Runtime prerequisites**
At least one of the following conditions must be true:

- Declare the `HIGH_SAMPLING_RATE_SENSORS` permission in your manifest.
- Request and be granted at least one of the following runtime permissions:
  - `BODY_SENSORS`
  - `ACTIVITY_RECOGNITION`

**Note:** The `BODY_SENSORS` runtime permission is subject to while-in-use restrictions. For this reason, you cannot create a `health` foreground service that uses body sensors while your app is in the background, with a few exceptions. For more information, see Restrictions on starting foreground services that need while-in-use permissions.

**Description**
Any long-running use cases to support apps in the fitness category such as exercise trackers.

## Location

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`location`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_LOCATION`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_LOCATION`

**Runtime prerequisites**
Request and be granted at least one of the following runtime permissions:

- `ACCESS_COARSE_LOCATION`
- `ACCESS_FINE_LOCATION`

**Note:** The location runtime permissions are subject to while-in-use restrictions. For this reason, you cannot create a `location` foreground service while your app is in the background, with a few exceptions. For more information, see Restrictions on starting foreground services that need while-in-use permissions.

**Description**
Long-running use cases that require location access, such as navigation and location sharing.

**Alternatives**
If your app needs to be triggered when the user reaches specific locations, consider using the geofence API instead.

## Media

**Foreground service type to declare in manifest under**
`android:foregroundServiceType`
`mediaPlayback`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_MEDIA_PLAYBACK`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_MEDIA_PLAYBACK`

**Runtime prerequisites**
None

**Description**
Continue audio or video playback from the background. Support Digital Video Recording (DVR) functionality on Android TV.

**Alternatives**
If you're showing picture-in-picture video, use Picture-in-Picture mode.

## Media projection

**Foreground service type to declare in manifest under**
`android:foregroundServiceType`
`mediaProjection`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_MEDIA_PROJECTION`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_MEDIA_PROJECTION`

**Runtime prerequisites**
Call the `createScreenCaptureIntent()` method before starting the foreground service. Doing so shows a permission notification to the user; the user must grant the permission before you can create the service.

After you have created the foreground service, you can call `MediaProjectionManager.getMediaProjection()`.

**Description**
Project content to non-primary display or external device using the `MediaProjection` APIs. This content doesn't have to be exclusively media content.

**Alternatives**
To stream media to another device, use the Google Cast SDK.

## Microphone

**Foreground service type to declare in manifest under**
`android:foregroundServiceType`

`microphone`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_MICROPHONE`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_MICROPHONE`

**Runtime prerequisites**
Request and be granted the `RECORD_AUDIO` runtime permission.

**Note:** The `RECORD_AUDIO` runtime permission is subject to while-in-use restrictions. For this reason, you cannot create a `microphone` foreground service while your app is in the background, with a few exceptions. For more information, see Restrictions on starting foreground services that need while-in-use permissions.

**Description**
Continue microphone capture from the background, such as voice recorders or communication apps.

## Phone call

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`phoneCall`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_PHONE_CALL`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_PHONE_CALL`

**Runtime prerequisites**
At least one of these conditions must be true:

> App has declared the `MANAGE_OWN_CALLS` permission in its manifest file.
>
> App is the default dialer app through the `ROLE_DIALER` role.

**Description**
Continue an ongoing call using the `ConnectionService` APIs.

**Alternatives**
If you need to make phone, video, or VoIP calls, consider using the `android.telecom` library.

Consider using `CallScreeningService` to screen calls.

## Remote messaging

## Short service

**Foreground service type to declare in manifest under `android:foregroundServiceType`**
`shortService`

**Permission to declare in your manifest**
None

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_SHORT_SERVICE`

**Runtime prerequisites**
None

**Description**
Quickly finish critical work that cannot be interrupted or postponed.

This type has some unique characteristics:

- Can only run for a short period of time (about 3 minutes).
- No support for sticky foreground services.
- Cannot start other foreground services.
- Doesn't require a type-specific permission, though it still requires the `FOREGROUND_SERVICE` permission.
- A `shortService` can only change to another service type if the app is currently eligible to start a new foreground service.
- A foreground service can change its type to `shortService` at any time, at which point the timeout period begins.

The timeout for shortService begins from the moment that `Service.startForeground()` is called. The app is expected to call `Service.stopSelf()` or `Service.stopForeground()` before the timeout occurs. Otherwise, the new `Service.onTimeout()` is called, giving apps a brief opportunity to call `stopSelf()` or `stopForeground()` to stop their service.

A short time after `Service.onTimeout()` is called, the app enters a cached state and is no longer considered to be in the foreground, unless the user is actively interacting with the app. A short time after the app is cached and the service has not stopped, the app receives an ANR. The ANR message mentions `FOREGROUND_SERVICE_TYPE_SHORT_SERVICE`. For these reasons, it's considered best practice to implement the `Service.onTimeout()` callback.

The `Service.onTimeout()` callback doesn't exist on Android 13 and lower. If the same service runs on such devices, it doesn't receive a timeout, nor does it ANR. Make sure that your service stops as soon as it finishes the processing task, even if it hasn't received the `Service.onTimeout()` callback yet.

It's important to note that if the timeout of the `shortService` is not respected, the app will ANR even if it has other valid foreground services or other app lifecycle processes running.

If an app is visible to the user or satisfies one of the exemptions that allow foreground services to be started from the background, calling `Service.StartForeground()` again with the `FOREGROUND_SERVICE_TYPE_SHORT_SERVICE` parameter extends the timeout by another 3 minutes. If the app isn't visible to the user and doesn't satisfy one of the exemptions, any attempt to start another foreground service, regardless of type, causes a `ForegroundServiceStartNotAllowedException`.

If a user disables battery optimization for your app, it's still affected by the timeout of shortService FGS.

If you start a foreground service that includes the `shortService` type and another foreground service type, the system ignores the `shortService` type declaration. However, the service must still adhere to the prerequisites of the other declared types. For more information, see the Foreground services documentation.

## Special use

**Foreground service type to declare in manifest under**
**`android:foregroundServiceType`**
`specialUse`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_SPECIAL_USE`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_SPECIAL_USE`

**Runtime prerequisites**
None

**Description**
Covers any valid foreground service use cases that aren't covered by the other foreground service types.

In addition to declaring the `FOREGROUND_SERVICE_TYPE_SPECIAL_USE` foreground service type, developers should declare use cases in the manifest. To do so, they specify the `<property>` element within the `<service>` element. These values and corresponding use cases are reviewed when you submit your app in the Google Play Console. The use cases you provide are free-form, and you should make sure to provide enough information to let the reviewer see why you need to use the `specialUse` type.

```
<service android:name="fooService" android:foregroundServiceType="specialUse">
  <property android:name="android.app.PROPERTY_SPECIAL_USE_FGS_SUBTYPE"
      android:value="explanation_for_special_use"/>
</service>
```

## System exempted

**Foreground service type to declare in manifest under**
**`android:foregroundServiceType`**
`systemExempted`

**Permission to declare in your manifest**
`FOREGROUND_SERVICE_SYSTEM_EXEMPTED`

**Constant to pass to `startForeground()`**
`FOREGROUND_SERVICE_TYPE_SYSTEM_EXEMPTED`

**Runtime prerequisites**
None

**Description**
Reserved for system applications and specific system integrations, to continue to use foreground services.

To use this type, an app must meet at least one of the following criteria:

- Device is in demo mode state
- App is a Device Owner
- App is a Profiler Owner
- Safety Apps that have the `ROLE_EMERGENCY` role
- Device Admin apps
- Apps holding `SCHEDULE_EXACT_ALARM` or `USE_EXACT_ALARM` permission
- VPN apps (configured using **Settings > Network & Internet > VPN**)

  Otherwise, declaring this type causes the system to throw a `ForegroundServiceTypeNotAllowedException`.

## Google Play policy enforcement for using foreground service types

If your app targets Android 14 or higher, you'll need to declare your app's foreground service types in the Play Console's app content page (**Policy > App content**). For more information on how to declare your foreground service types in Play Console, see Understanding foreground service and full-screen intent requirements.