Intents and Intent Filters

An __Intent__ (https://developer.android.com/reference/android/content/Intent.html) is a messaging object you can use to request an action from another app component (https://developer.android.com/guide/components/fundamentals.html#Components). Although intents facilitate communication between components in several ways, there are three fundamental use cases:

- **Starting an activity**

  An __Activity__ (https://developer.android.com/reference/android/app/Activity.html) represents a single screen in an app. You can start a new instance of an __Activity__ (https://developer.android.com/reference/android/app/Activity.html) by passing an __Intent__ (https://developer.android.com/reference/android/content/Intent.html) to __startActivity()__ (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) . The __Intent__ (https://developer.android.com/reference/android/content/Intent.html) describes the activity to start and carries any necessary data.

  If you want to receive a result from the activity when it finishes, call __startActivityForResult()__ (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) . Your activity receives the result as a separate __Intent__ (https://developer.android.com/reference/android/content/Intent.html) object in your activity's __onActivityResult()__ (https://developer.android.com/reference/android/app/Activity.html#onActivityResult(int, int, android.content.Intent)) callback. For more information, see the Activities (https://developer.android.com/guide/components/activities.html) guide.

- **Starting a service**

  A __Service__ (https://developer.android.com/reference/android/app/Service.html) is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with __JobScheduler__ (https://developer.android.com/reference/android/app/job/JobScheduler.html). For more information about __JobScheduler__ (https://developer.android.com/reference/android/app/job/JobScheduler.html), see its __API-reference documentation__ (https://developer.android.com/reference/android/app/job/JobScheduler.html).

For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the `Service` (https://developer.android.com/reference/android/app/Service.html) class. You can start a service to perform a one-time operation (such as downloading a file) by passing an `Intent` (https://developer.android.com/reference/android/content/Intent.html) to `startService()` (https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)) . The `Intent` (https://developer.android.com/reference/android/content/Intent.html) describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an `Intent` (https://developer.android.com/reference/android/content/Intent.html) to `bindService()` (https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) . For more information, see the Services (https://developer.android.com/guide/components/services.html) guide.

- **Delivering a broadcast**

  A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an `Intent` (https://developer.android.com/reference/android/content/Intent.html) to `sendBroadcast()` (https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)) or `sendOrderedBroadcast()` (https://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)) .

The rest of this page explains how intents work and how to use them. For related information, see Interacting with Other Apps (https://developer.android.com/training/basics/intents/index.html) and Sharing Content (https://developer.android.com/training/sharing/index.html).

Intent types

There are two types of intents:

- **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the

class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.

- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Figure 1 shows how an intent is used when starting an activity. When the **Intent** (https://developer.android.com/reference/android/content/Intent.html) object names a specific activity component explicitly, the system immediately starts that component.
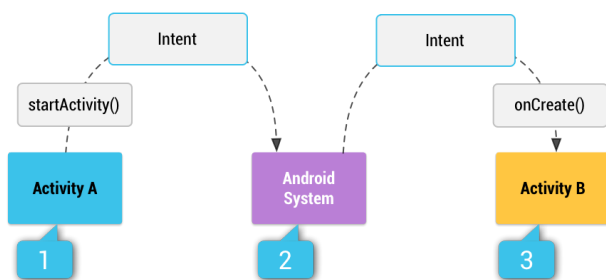

Figure 1. How an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an **Intent** (https://developer.android.com/reference/android/content/Intent.html) with an action description and passes it to **startActivity()** (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) . **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its **onCreate()** (https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)) method and passing it the **Intent** (https://developer.android.com/reference/android/content/Intent.html) .

When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file (https://developer.android.com/guide/topics/manifest/manifest-intro.html) of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the `Intent` (https://developer.android.com/reference/android/content/Intent.html) object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

Caution: To ensure that your app is secure, always use an explicit intent when starting a `Service` (https://developer.android.com/reference/android/app/Service.html) and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call `bindService()` (https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) with an implicit intent.

Building an intent

An `Intent` (https://developer.android.com/reference/android/content/Intent.html) object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

The primary information contained in an `Intent` (https://developer.android.com/reference/android/content/Intent.html) is the following:

Component name

> The name of the component to start.
> This is optional, but it's the critical piece of information that makes an intent *explicit*, meaning that the intent should be delivered only to the app component defined by the component name. Without a component name, the intent is *implicit* and the system

decides which component should receive the intent based on the other intent information (such as the action, data, and category—described below). If you need to start a specific component in your app, you should specify the component name.

Note: When starting a **Service** (https://developer.android.com/reference/android/app/Service.html), *always specify the component name*. Otherwise, you cannot be certain what service will respond to the intent, and the user cannot see which service starts.

This field of the **Intent** (https://developer.android.com/reference/android/content/Intent.html) is a **ComponentName** (https://developer.android.com/reference/android/content/ComponentName.html) object, which you can specify using a fully qualified class name of the target component, including the package name of the app, for example, `com.example.ExampleActivity`. You can set the component name with **setComponent()** (https://developer.android.com/reference/android/content/Intent.html#setComponent(android.content.ComponentName)) , **setClass()** (https://developer.android.com/reference/android/content/Intent.html#setClass(android.content.Context, java.lang.Class<?>)) , **setClassName()** (https://developer.android.com/reference/android/content/Intent.html#setClassName(java.lang.String, java.lang.String)) , or with the **Intent** (https://developer.android.com/reference/android/content/Intent.html) constructor.

Action

A string that specifies the generic action to perform (such as *view* or *pick*).

In the case of a broadcast intent, this is the action that took place and is being reported. The action largely determines how the rest of the intent is structured— particularly the information that is contained in the data and extras.

You can specify your own actions for use by intents within your app (or for use by other apps to invoke components in your app), but you usually specify action constants defined by the **Intent** (https://developer.android.com/reference/android/content/Intent.html) class or other framework classes. Here are some common actions for starting an activity:

ACTION_VIEW
(https://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

> Use this action in an intent with `startActivity()`
> (https://developer.android.com/reference/android/content/Context.html#startActivity(andr
> oid.content.Intent))
> when you have some information that an activity can show to the user, such as a
> photo to view in a gallery app, or an address to view in a map app.

ACTION_SEND
(https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND)

> Also known as the *share* intent, you should use this in an intent with
> `startActivity()`
> (https://developer.android.com/reference/android/content/Context.html#startActivity(andr
> oid.content.Intent))
> when you have some data that the user can share through another app, such as
> an email app or social sharing app.

See the **Intent** (https://developer.android.com/reference/android/content/Intent.html) class
reference for more constants that define generic actions. Other actions are defined
elsewhere in the Android framework, such as in **Settings**
(https://developer.android.com/reference/android/provider/Settings.html) for actions that
open specific screens in the system's Settings app.

You can specify the action for an intent with **setAction()**
(https://developer.android.com/reference/android/content/Intent.html#setAction(java.lang.String
))
or with an **Intent** (https://developer.android.com/reference/android/content/Intent.html)
constructor.

If you define your own actions, be sure to include your app's package name as a prefix,
as shown in the following example:

```
static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRA
```

Data

The URI (a **Uri** (https://developer.android.com/reference/android/net/Uri.html) object) that
references the data to be acted on and/or the MIME type of that data. The type of
data supplied is generally dictated by the intent's action. For example, if the action is
ACTION_EDIT
(https://developer.android.com/reference/android/content/Intent.html#ACTION_EDIT), the data
should contain the URI of the document to edit.

When creating an intent, it's often important to specify the type of data (its MIME type) in addition to its URI. For example, an activity that's able to display images probably won't be able to play an audio file, even though the URI formats could be similar. Specifying the MIME type of your data helps the Android system find the best component to receive your intent. However, the MIME type can sometimes be inferred from the URI—particularly when the data is a `content:` URI. A `content:` URI indicates the data is located on the device and controlled by a <u>ContentProvider</u> (https://developer.android.com/reference/android/content/ContentProvider.html), which makes the data MIME type visible to the system.

To set only the data URI, call <u>setData()</u> (https://developer.android.com/reference/android/content/Intent.html#setData(android.net.Uri)). To set only the MIME type, call <u>setType()</u> (https://developer.android.com/reference/android/content/Intent.html#setType(java.lang.String)) . If necessary, you can set both explicitly with <u>setDataAndType()</u> (https://developer.android.com/reference/android/content/Intent.html#setDataAndType(android. net.Uri, java.lang.String)) .

> ⓘ **Caution:** If you want to set both the URI and MIME type, *don't* call <u>setData()</u> (https://developer.android.com/reference/android/content/Intent.html#setData(android.net.Uri)) and <u>setType()</u> (https://developer.android.com/reference/android/content/Intent.html#setType(java.lang.String)) because they each nullify the value of the other. Always use <u>setDataAndType()</u> (https://developer.android.com/reference/android/content/Intent.html#setDataAndType(android. net.Uri, java.lang.String)) to set both URI and MIME type.

Category

A string containing additional information about the kind of component that should handle the intent. Any number of category descriptions can be placed in an intent, but most intents do not require a category. Here are some common categories:

<u>CATEGORY_BROWSABLE</u> (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_BROWSABLE)

> The target activity allows itself to be started by a web browser to display data referenced by a link, such as an image or an e-mail message.

`CATEGORY_LAUNCHER`
(https://developer.android.com/reference/android/content/Intent.html#CATEGORY_LAUNCHER)

> The activity is the initial activity of a task and is listed in the system's application launcher.

See the `Intent` (https://developer.android.com/reference/android/content/Intent.html) class description for the full list of categories.

You can specify a category with `addCategory()` (https://developer.android.com/reference/android/content/Intent.html#addCategory(java.lang.String))
.

These properties listed above (component name, action, data, and category) represent the defining characteristics of an intent. By reading these properties, the Android system is able to resolve which app component it should start. However, an intent can carry additional information that does not affect how it is resolved to an app component. An intent can also supply the following information:

Extras

> Key-value pairs that carry additional information required to accomplish the requested action. Just as some actions use particular kinds of data URIs, some actions also use particular extras.
> You can add extra data with various `putExtra()` (https://developer.android.com/reference/android/content/Intent.html#putExtra(java.lang.String, android.os.Bundle))
> methods, each accepting two parameters: the key name and the value. You can also create a `Bundle` (https://developer.android.com/reference/android/os/Bundle.html) object with all the extra data, then insert the `Bundle` (https://developer.android.com/reference/android/os/Bundle.html) in the `Intent` (https://developer.android.com/reference/android/content/Intent.html) with `putExtras()` (https://developer.android.com/reference/android/content/Intent.html#putExtras(android.content.Intent))
> .

> For example, when creating an intent to send an email with `ACTION_SEND` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND), you can specify the *to* recipient with the `EXTRA_EMAIL` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_EMAIL) key, and specify the *subject* with the `EXTRA_SUBJECT` (https://developer.android.com/reference/android/content/Intent.html#EXTRA_SUBJECT) key.

The **Intent** (https://developer.android.com/reference/android/content/Intent.html) class specifies many `EXTRA_*` constants for standardized data types. If you need to declare your own extra keys (for intents that your app receives), be sure to include your app's package name as a prefix, as shown in the following example:

```
static final String EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS";
```

> ⚠ **Caution**: Do not use **Parcelable** (https://developer.android.com/reference/android/os/Parcelable.html) or **Serializable** (https://developer.android.com/reference/java/io/Serializable.html) data when sending an intent that you expect another app to receive. If an app attempts to access data in a **Bundle** (https://developer.android.com/reference/android/os/Bundle.html) object but does not have access to the parceled or serialized class, the system raises a **RuntimeException** (https://developer.android.com/reference/java/lang/RuntimeException.html).

Flags

Flags are defined in the **Intent** (https://developer.android.com/reference/android/content/Intent.html) class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity (for example, which task (https://developer.android.com/guide/components/tasks-and-back-stack.html) the activity should belong to) and how to treat it after it's launched (for example, whether it belongs in the list of recent activities).

For more information, see the **setFlags()** (https://developer.android.com/reference/android/content/Intent.html#setFlags(int)) method.

Example explicit intent

An explicit intent is one that you use to launch a specific app component, such as a particular activity or service in your app. To create an explicit intent, define the component name for the **Intent** (https://developer.android.com/reference/android/content/Intent.html) object —all other intent properties are optional.

For example, if you built a service in your app, named `DownloadService`, designed to download a file from the web, you can start it with the following code:

```
// Executed in an Activity, so 'this' is the Context
(https://developer.android.com/reference/android/content/Context.html)
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
```

```
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse (https://developer.android.com/reference/android/net/Uri.ht
startService(downloadIntent);
```

The Intent(Context, Class)
(https://developer.android.com/reference/android/content/Intent.html#Intent(android.content.Context,
java.lang.Class<?>))
constructor supplies the app Context
(https://developer.android.com/reference/android/content/Context.html) and the component a
Class (https://developer.android.com/reference/java/lang/Class.html) object. As such, this intent
explicitly starts the DownloadService class in the app.

For more information about building and starting a service, see the Services
(https://developer.android.com/guide/components/services.html) guide.

Example implicit intent

An implicit intent specifies an action that can invoke any app on the device able to perform
the action. Using an implicit intent is useful when your app cannot perform the action, but
other apps probably can and you'd like the user to pick which app to use.

For example, if you have content that you want the user to share with other people, create
an intent with the ACTION_SEND
(https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) action and
add extras that specify the content to share. When you call startActivity()
(https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.I
ntent))
with that intent, the user can pick an app through which to share the content.

Caution: It's possible that a user won't have *any* apps that handle the implicit intent you send to
startActivity()
(https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.I
ntent))
. Or, an app may be inaccessible because of profile restrictions or settings put into place by an
administrator. If that happens, the call fails and your app crashes. To verify that an activity will receive
the intent, call resolveActivity()
(https://developer.android.com/reference/android/content/Intent.html#resolveActivity(android.content.
pm.PackageManager))
on your Intent (https://developer.android.com/reference/android/content/Intent.html) object. If the
result is non-null, there is at least one app that can handle the intent and it's safe to call
startActivity()

. If the result is null, do not use the intent and, if possible, you should disable the feature that issues the intent. The following example shows how to verify that the intent resolves to an activity. This example doesn't use a URI, but the intent's data type is declared to specify the content carried by the extras.

```java
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

When **startActivity()**
 (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent))
is called, the system examines all of the installed apps to determine which ones can handle this kind of intent (an intent with the **ACTION_SEND**
 (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) action and that carries "text/plain" data). If there's only one app that can handle it, that app opens immediately and is given the intent. If multiple activities accept the intent, the system displays a dialog such as the one shown in Figure 2, so the user can pick which app to use.

More information about launching other apps is also provided in Sending the User to Another App (https://developer.android.com/training/basics/intents/sending.html).
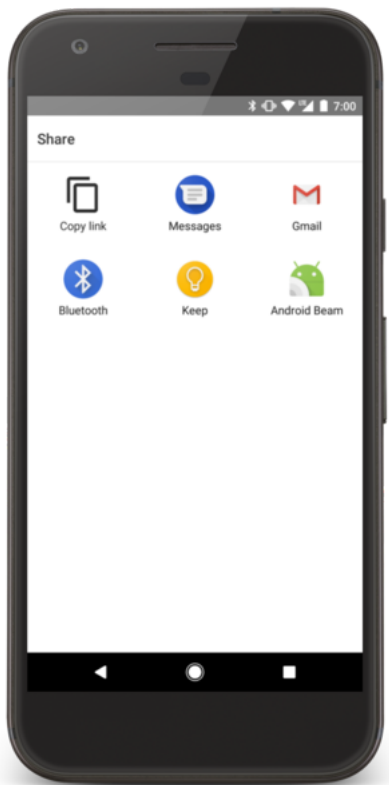
Figure 2. A chooser dialog.

Forcing an app chooser

When there is more than one app that responds to your implicit intent, the user can select which app to use and make that app the default choice for the action. The ability to select a default is helpful when performing an action for which the user probably wants to use the same app every time, such as when opening a web page (users often prefer just one web browser).

However, if multiple apps can respond to the intent and the user might want to use a different app each time, you should explicitly show a chooser dialog. The chooser dialog asks the user to select which app to use for the action (the user cannot select a default app for the action). For example, when your app performs "share" with the `ACTION_SEND` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) action, users may want to share using a different app depending on their current situation, so you should always use the chooser dialog, as shown in Figure 2.

To show the chooser, create an `Intent` (https://developer.android.com/reference/android/content/Intent.html) using `createChooser()` (https://developer.android.com/reference/android/content/Intent.html#createChooser(android.content. Intent, java.lang.CharSequence)) and pass it to `startActivity()`.

(https://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent))
, as shown in the following example. This example displays a dialog with a list of apps that respond to the intent passed to the <u>**createChooser()**</u>
(https://developer.android.com/reference/android/content/Intent.html#createChooser(android.content.Intent, java.lang.CharSequence))
method and uses the supplied text as the dialog title.

```java
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

Receiving an implicit intent

To advertise which implicit intents your app can receive, declare one or more intent filters for each of your app components with an <u>**<intent-filter>**</u>
(https://developer.android.com/guide/topics/manifest/intent-filter-element.html) element in your manifest file (https://developer.android.com/guide/topics/manifest/manifest-intro.html). Each intent filter specifies the type of intents it accepts based on the intent's action, data, and category. The system delivers an implicit intent to your app component only if the intent can pass through one of your intent filters.

Note: An explicit intent is always delivered to its target, regardless of any intent filters the component declares.

An app component should declare separate filters for each unique job it can do. For example, one activity in an image gallery app may have two filters: one filter to view an image, and another filter to edit an image. When the activity starts, it inspects the <u>**Intent**</u>
(https://developer.android.com/reference/android/content/Intent.html) and decides how to behave based on the information in the <u>**Intent**</u>

(https://developer.android.com/reference/android/content/Intent.html) (such as to show the editor controls or not).

Each intent filter is defined by an `<intent-filter>` (https://developer.android.com/guide/topics/manifest/intent-filter-element.html) element in the app's manifest file, nested in the corresponding app component (such as an `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html) element). Inside the `<intent-filter>` (https://developer.android.com/guide/topics/manifest/intent-filter-element.html), you can specify the type of intents to accept using one or more of these three elements:

`<action>` (https://developer.android.com/guide/topics/manifest/action-element.html)

Declares the intent action accepted, in the `name` attribute. The value must be the literal string value of an action, not the class constant.

`<data>` (https://developer.android.com/guide/topics/manifest/data-element.html)

Declares the type of data accepted, using one or more attributes that specify various aspects of the data URI (`scheme`, `host`, `port`, `path`) and MIME type.

`<category>` (https://developer.android.com/guide/topics/manifest/category-element.html)

Declares the intent category accepted, in the `name` attribute. The value must be the literal string value of an action, not the class constant.

> ⭐ **Note:** To receive implicit intents, you *must include* the `CATEGORY_DEFAULT` (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_DEFAULT) category in the intent filter. The methods `startActivity()` (https://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent)) and `startActivityForResult()` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) treat all intents as if they declared the `CATEGORY_DEFAULT` (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_DEFAULT) category. If you do not declare this category in your intent filter, no implicit intents will resolve to your activity.

For example, here's an activity declaration with an intent filter to receive an `ACTION_SEND` (https://developer.android.com/reference/android/content/Intent.html#ACTION_SEND) intent when the data type is text:

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

You can create a filter that includes more than one instance of <u>**<action>**</u>
 (https://developer.android.com/guide/topics/manifest/action-element.html), <u>**<data>**</u>
 (https://developer.android.com/guide/topics/manifest/data-element.html), or <u>**<category>**</u>
 (https://developer.android.com/guide/topics/manifest/category-element.html). If you do, you need
to be certain that the component can handle any and all combinations of those filter
elements.

When you want to handle multiple kinds of intents, but only in specific combinations of
action, data, and category type, then you need to create multiple intent filters.

Restricting access to components

Using an intent filter is not a secure
way to prevent other apps from
starting your components.
Although intent filters restrict a
component to respond to only
certain kinds of implicit intents,
another app can potentially start
your app component by using an
explicit intent if the developer
determines your component
names. If it's important that *only
your own app* is able to start one of
your components, do not declare
intent filters in your manifest.
Instead, set the <u>**exported**</u>
 (https://developer.android.com/guide/t
opics/manifest/activity-
element.html#exported)

attribute to "`false`" for that
component.

An implicit intent is tested against a filter by comparing the intent to each of the three elements. To be delivered to the component, the intent must pass all three tests. If it fails to match even one of them, the Android system won't deliver the intent to the component. However, because a component may have multiple intent filters, an intent that does not pass through one of a component's filters might make it through on another filter. More information about how the system resolves intents is provided in the section below about Intent Resolution (#Resolution).

Caution: To avoid inadvertently running a different app's **Service** (https://developer.android.com/reference/android/app/Service.html), always use an explicit intent to start your own service and do not declare intent filters for your service.

Note: For all activities, you must declare your intent filters in the manifest file. However, filters for broadcast receivers can be registered dynamically by calling **registerReceiver()** (https://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter, java.lang.String, android.os.Handler)) . You can then unregister the receiver with **unregisterReceiver()** (https://developer.android.com/reference/android/content/Context.html#unregisterReceiver(android.content.BroadcastReceiver)) . Doing so allows your app to listen for specific broadcasts during only a specified period of time while your app is running.

Example filters

To demonstrate some of the intent filter behaviors, here is an example from the manifest file of a social-sharing app:

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
```

```
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

The first activity, `MainActivity`, is the app's main entry point—the activity that opens when the user initially launches the app with the launcher icon:

- The `ACTION_MAIN`
  (https://developer.android.com/reference/android/content/Intent.html#ACTION_MAIN) action indicates this is the main entry point and does not expect any intent data.

- The `CATEGORY_LAUNCHER`
  (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_LAUNCHER) category indicates that this activity's icon should be placed in the system's app launcher. If the `<activity>`
  (https://developer.android.com/guide/topics/manifest/activity-element.html) element does not specify an icon with `icon`, then the system uses the icon from the `<application>`
  (https://developer.android.com/guide/topics/manifest/application-element.html) element.

These two must be paired together in order for the activity to appear in the app launcher.

The second activity, `ShareActivity`, is intended to facilitate sharing text and media content. Although users might enter this activity by navigating to it from `MainActivity`, they can also enter `ShareActivity` directly from another app that issues an implicit intent matching one of the two intent filters.

Note: The MIME type, `application/vnd.google.panorama360+jpg`
 (https://developers.google.com/panorama/android/), is a special data type that specifies panoramic photos, which you can handle with the Google panorama
 (https://developer.android.com/reference/com/google/android/gms/panorama/package-summary.html)
APIs.

Using a pending intent

A `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) object is a wrapper around an `Intent` (https://developer.android.com/reference/android/content/Intent.html) object. The primary purpose of a `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) is to grant permission to a foreign application to use the contained `Intent` (https://developer.android.com/reference/android/content/Intent.html) as if it were executed from your app's own process.

Major use cases for a pending intent include the following:

- Declaring an intent to be executed when the user performs an action with your Notification (https://developer.android.com/guide/topics/ui/notifiers/notifications.html) (the Android system's `NotificationManager` (https://developer.android.com/reference/android/app/NotificationManager.html) executes the `Intent` (https://developer.android.com/reference/android/content/Intent.html)).

- Declaring an intent to be executed when the user performs an action with your App Widget (https://developer.android.com/guide/topics/appwidgets/index.html) (the Home screen app executes the `Intent` (https://developer.android.com/reference/android/content/Intent.html)).

- Declaring an intent to be executed at a specified future time (the Android system's `AlarmManager` (https://developer.android.com/reference/android/app/AlarmManager.html) executes the `Intent` (https://developer.android.com/reference/android/content/Intent.html)).

Just as each `Intent` (https://developer.android.com/reference/android/content/Intent.html) object is designed to be handled by a specific type of app component (either an `Activity` (https://developer.android.com/reference/android/app/Activity.html), a `Service` (https://developer.android.com/reference/android/app/Service.html), or a `BroadcastReceiver` (https://developer.android.com/reference/android/content/BroadcastReceiver.html)), so too must a `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) be created with the same consideration. When using a pending intent, your app doesn't execute the intent with a call such as `startActivity()` (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) . Instead, you must declare the intended component type when you create the `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) by calling the respective creator method:

- `PendingIntent.getActivity()` (https://developer.android.com/reference/android/app/PendingIntent.html#getActivity(android.content.Context, int, android.content.Intent, int))

  for an `Intent` (https://developer.android.com/reference/android/content/Intent.html) that starts an `Activity` (https://developer.android.com/reference/android/app/Activity.html).

- `PendingIntent.getService()` (https://developer.android.com/reference/android/app/PendingIntent.html#getService(android.content.Context, int, android.content.Intent, int))

  for an `Intent` (https://developer.android.com/reference/android/content/Intent.html) that starts a `Service` (https://developer.android.com/reference/android/app/Service.html).

- `PendingIntent.getBroadcast()` (https://developer.android.com/reference/android/app/PendingIntent.html#getBroadcast(android.content.Context, int, android.content.Intent, int))

  for a `Intent` (https://developer.android.com/reference/android/content/Intent.html) that starts an `BroadcastReceiver` (https://developer.android.com/reference/android/content/BroadcastReceiver.html).

Unless your app is *receiving* pending intents from other apps, the above methods to create a `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) are probably the only `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) methods you'll ever need.

Each method takes the current app `Context` (https://developer.android.com/reference/android/content/Context.html), the `Intent` (https://developer.android.com/reference/android/content/Intent.html) you want to wrap, and one or more flags that specify how the intent should be used (such as whether the intent can be used more than once).

For more information about using pending intents, see the documentation for each of the respective use cases, such as in the Notifications (https://developer.android.com/guide/topics/ui/notifiers/notifications.html) and App Widgets (https://developer.android.com/guide/topics/appwidgets/index.html) API guides.

Intent resolution

When the system receives an implicit intent to start an activity, it searches for the best activity for the intent by comparing it to intent filters based on three aspects:

- Action.

- Data (both URI and data type).

- Category.

The following sections describe how intents are matched to the appropriate components according to the intent filter declaration in an app's manifest file.

Action test

To specify accepted intent actions, an intent filter can declare zero or more **<action>** (https://developer.android.com/guide/topics/manifest/action-element.html) elements, as shown in the following example:

```
<intent-filter>
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.VIEW" />
    ...
</intent-filter>
```

To pass this filter, the action specified in the **Intent** (https://developer.android.com/reference/android/content/Intent.html) must match one of the actions listed in the filter.

If the filter does not list any actions, there is nothing for an intent to match, so all intents fail the test. However, if an **Intent** (https://developer.android.com/reference/android/content/Intent.html) does not specify an action, it passes the test as long as the filter contains at least one action.

Category test

To specify accepted intent categories, an intent filter can declare zero or more **<category>** (https://developer.android.com/guide/topics/manifest/category-element.html) elements, as shown in the following example:

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    ...
</intent-filter>
```

For an intent to pass the category test, every category in the **Intent** (https://developer.android.com/reference/android/content/Intent.html) must match a category in the filter. The reverse is not necessary—the intent filter may declare more categories than

are specified in the **Intent** (https://developer.android.com/reference/android/content/Intent.html) and the **Intent** (https://developer.android.com/reference/android/content/Intent.html) still passes. Therefore, an intent with no categories always passes this test, regardless of what categories are declared in the filter.

Note: Android automatically applies the **CATEGORY_DEFAULT** (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_DEFAULT) category to all implicit intents passed to **startActivity()** (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) and **startActivityForResult()** (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)). If you want your activity to receive implicit intents, it must include a category for `"android.intent.category.DEFAULT"` in its intent filters, as shown in the previous `<intent-filter>` example.

Data test

To specify accepted intent data, an intent filter can declare zero or more **<data>** (https://developer.android.com/guide/topics/manifest/data-element.html) elements, as shown in the following example:

```
<intent-filter>
    <data android:mimeType="video/mpeg" android:scheme="http" ... />
    <data android:mimeType="audio/mpeg" android:scheme="http" ... />
    ...
</intent-filter>
```

Each **<data>** (https://developer.android.com/guide/topics/manifest/data-element.html) element can specify a URI structure and a data type (MIME media type). Each part of the URI is a separate attribute: `scheme`, `host`, `port`, and `path`:

`<scheme>://<host>:<port>/<path>`

The following example shows possible values for these attributes:

`content://com.example.project:200/folder/subfolder/etc`

In this URI, the scheme is `content`, the host is `com.example.project`, the port is `200`, and the path is `folder/subfolder/etc`.

Each of these attributes is optional in a <u>**`<data>`**</u>
 (https://developer.android.com/guide/topics/manifest/data-element.html) element, but there are
linear dependencies:

- If a scheme is not specified, the host is ignored.

- If a host is not specified, the port is ignored.

- If both the scheme and host are not specified, the path is ignored.

When the URI in an intent is compared to a URI specification in a filter, it's compared only to
the parts of the URI included in the filter. For example:

- If a filter specifies only a scheme, all URIs with that scheme match the filter.

- If a filter specifies a scheme and an authority but no path, all URIs with the same
  scheme and authority pass the filter, regardless of their paths.

- If a filter specifies a scheme, an authority, and a path, only URIs with the same
  scheme, authority, and path pass the filter.

Note: A path specification can contain a wildcard asterisk (*) to require only a partial match of the path
name.

The data test compares both the URI and the MIME type in the intent to a URI and MIME
type specified in the filter. The rules are as follows:

1. An intent that contains neither a URI nor a MIME type passes the test only if the filter
   does not specify any URIs or MIME types.

2. An intent that contains a URI but no MIME type (neither explicit nor inferable from the
   URI) passes the test only if its URI matches the filter's URI format and the filter
   likewise does not specify a MIME type.

3. An intent that contains a MIME type but not a URI passes the test only if the filter lists
   the same MIME type and does not specify a URI format.

4. An intent that contains both a URI and a MIME type (either explicit or inferable from
   the URI) passes the MIME type part of the test only if that type matches a type listed
   in the filter. It passes the URI part of the test either if its URI matches a URI in the filter
   or if it has a `content:` or `file:` URI and the filter does not specify a URI. In other
   words, a component is presumed to support `content:` and `file:` data if its filter lists
   only a MIME type.

This last rule, rule (d), reflects the expectation that components are able to get local data
from a file or content provider. Therefore, their filters can list just a data type and don't need

to explicitly name the `content:` and `file:` schemes. The following example shows a typical case in which a <u>**<data>**</u> (https://developer.android.com/guide/topics/manifest/data-element.html) element tells Android that the component can get image data from a content provider and display it:

```
<intent-filter>
    <data android:mimeType="image/*" />
    ...
</intent-filter>
```

Filters that specify a data type but not a URI are perhaps the most common because most available data is dispensed by content providers.

Another common configuration is a filter with a scheme and a data type. For example, a <u>**<data>**</u> (https://developer.android.com/guide/topics/manifest/data-element.html) element like the following tells Android that the component can retrieve video data from the network in order to perform the action:

```
<intent-filter>
    <data android:scheme="http" android:type="video/*" />
    ...
</intent-filter>
```

Intent matching

Intents are matched against intent filters not only to discover a target component to activate, but also to discover something about the set of components on the device. For example, the Home app populates the app launcher by finding all the activities with intent filters that specify the <u>`ACTION_MAIN`</u> (https://developer.android.com/reference/android/content/Intent.html#ACTION_MAIN) action and <u>`CATEGORY_LAUNCHER`</u> (https://developer.android.com/reference/android/content/Intent.html#CATEGORY_LAUNCHER) category. A match is only successful if the actions and categories in the Intent match against the filter, as described in the documentation for the <u>`IntentFilter`</u> (https://developer.android.com/reference/android/content/IntentFilter.html) class.

Your application can use intent matching in a manner similar to what the Home app does. The <u>`PackageManager`</u> (https://developer.android.com/reference/android/content/pm/PackageManager.html) has a set of `query...()` methods that return all components that can accept a particular intent and a similar series of `resolve...()` methods that determine the best component to respond to an intent. For example, <u>`queryIntentActivities()`</u>

(https://developer.android.com/reference/android/content/pm/PackageManager.html#queryIntentActivities(android.content.Intent, int))

returns a list of all activities that can perform the intent passed as an argument, and `queryIntentServices()` (https://developer.android.com/reference/android/content/pm/PackageManager.html#queryIntentServices(android.content.Intent, int))

returns a similar list of services. Neither method activates the components; they just list the ones that can respond. There's a similar method, `queryBroadcastReceivers()` (https://developer.android.com/reference/android/content/pm/PackageManager.html#queryBroadcastReceivers(android.content.Intent, int))

, for broadcast receivers.

Twitter
Follow @AndroidDev on
Twitter

Google+
Follow Android Developers on
Google+

YouTube
Check out Android Developers
on YouTube