App Components

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior.

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of app components:

**Activities**

An *activity* represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of [Activity](https://developer.android.com/reference/android/app/Activity.html) and you can learn more about it in the [Activities](https://developer.android.com/guide/components/activities.html) developer guide.

**Services**

A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of [Service](https://developer.android.com/reference/android/app/Service.html) and you can learn more about it in the [Services](https://developer.android.com/guide/components/services.html) developer guide.

**Content providers**

A *content provider* manages a shared set of app data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as [ContactsContract.Data](https://developer.android.com/reference/android/provider/ContactsContract.Data.html)) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the [Note Pad](https://developer.android.com/resources/samples/NotePad/index.html) sample app uses a content provider to save notes.

A content provider is implemented as a subclass of [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the [Content Providers](https://developer.android.com/guide/topics/providers/content-providers.html) developer guide.

**Broadcast receivers**

A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may [create a status bar notification](https://developer.android.com/guide/topics/ui/notifiers/notifications.html) to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html) and each broadcast is delivered as an [Intent](https://developer.android.com/reference/android/content/Intent.html) object. For more information, see the [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html) class.

A unique aspect of the Android system design is that any app can start another app’s component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main()function, for example).

Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. The Android system, however, can. So, to activate a component in another app, you must deliver a message to the system that specifies your *intent* to start a particular component. The system then activates the component for you.

Activating Components

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or another.

An intent is created with an [Intent](https://developer.android.com/reference/android/content/Intent.html) object, which defines a message to activate either a specific component or a specific *type* of component—an intent can be either explicit or implicit, respectively.

For activities and services, an intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an [Intent](https://developer.android.com/reference/android/content/Intent.html) (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html). The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html) object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activating each type of component:

* You can start an activity (or give it something new to do) 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)) or [startActivityForResult()](https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)) (when you want the activity to return a result).
* You can start a service (or give new instructions to an ongoing service) 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)). Or you can bind to the service 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)).
* You can initiate a broadcast by passing an [Intent](https://developer.android.com/reference/android/content/Intent.html) to methods like [sendBroadcast()](https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)), [sendOrderedBroadcast()](https://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)), or [sendStickyBroadcast()](https://developer.android.com/reference/android/content/Context.html#sendStickyBroadcast(android.content.Intent)).
* You can perform a query to a content provider by calling [query()](https://developer.android.com/reference/android/content/ContentProvider.html#query(android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String)) on a [ContentResolver](https://developer.android.com/reference/android/content/ContentResolver.html).

For more information about using intents, see the [Intents and Intent Filters](https://developer.android.com/guide/components/intents-filters.html) document. More information about activating specific components is also provided in the following documents: [Activities](https://developer.android.com/guide/components/activities.html), [Services](https://developer.android.com/guide/components/services.html), [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html) and [Content Providers](https://developer.android.com/guide/topics/providers/content-providers.html).