Intents and Intent Filters

Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called *intents*. Intent messaging is a facility for late run-time binding between components in the same or different applications. The intent itself, an `Intent` object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced. There are separate mechanisms for delivering intents to each type of component:

- An Intent object is passed to `Context.startActivity()` or `Activity.startActivityForResult()` to launch an activity or get an existing activity to do something new. (It can also be passed to `Activity.setResult()` to return information to the activity that called `startActivityForResult()`.)
- An Intent object is passed to `Context.startService()` to initiate a service or deliver new instructions to an ongoing service. Similarly, an intent can be passed to `Context.bindService()` to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running.
- Intent objects passed to any of the broadcast methods (such as `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, or `Context.sendStickyBroadcast()`) are delivered to all interested broadcast receivers. Many kinds of broadcasts originate in system code.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to `startActivity()` is delivered only to an activity, never to a service or broadcast receiver, and so on.

This document begins with a description of Intent objects. It then describes the rules Android uses to map intents to components — how it resolves which component should receive an intent message. For intents that don't explicitly name a target component, this process involves testing the Intent object against *intent filters* associated with potential targets.

Intent Objects

An `Intent` object is a bundle of information. It contains information of interest to the component that receives the intent (such as the action to be taken and the data to act on) plus information of interest to the Android system (such as the category of component that should handle the intent and instructions on how to launch a target activity). Principally, it can contain the following:

Component name

The name of the component that should handle the intent. This field is a `ComponentName` object — a combination of the fully qualified class name of the target component (for example "`com.example.project.app.FreneticActivity`") and the package name set in the manifest file of the application where the component resides (for example, "`com.example.project`"). The package part of the component name and the package name set in the manifest do not necessarily have to match.

The component name is optional. If it is set, the Intent object is delivered to an instance of the designated class. If it is not set, Android uses other information in the Intent object to locate a suitable target — see Intent Resolution, later in this document.

The component name is set by `setComponent()`, `setClass()`, or `setClassName()` and read by `getComponent()`.

Action

A string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The Intent class defines a number of action constants, including these:

Constant	Target component	Action
`ACTION_CALL`	activity	Initiate a phone call.
`ACTION_EDIT`	activity	Display data for the user to edit.
`ACTION_MAIN`	activity	Start up as the initial activity of a task, with no data input and no returned output.
`ACTION_SYNC`	activity	Synchronize data on a server with data on the mobile device.
`ACTION_BATTERY_LOW`	broadcast	A warning that the battery is low.

	receiver	
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	The setting for the time zone has changed.

See the Intent class description for a list of pre-defined constants for generic actions. Other actions are defined elsewhere in the Android API. You can also define your own action strings for activating the components in your application. Those you invent should include the application package as a prefix — for example: "com.example.project.SHOW_COLOR".

The action largely determines how the rest of the intent is structured — particularly the data and extras fields — much as a method name determines a set of arguments and a return value. For this reason, it's a good idea to use action names that are as specific as possible, and to couple them tightly to the other fields of the intent. In other words, instead of defining an action in isolation, define an entire protocol for the Intent objects your components can handle.

The action in an Intent object is set by the setAction() method and read by getAction().

Data

The URI of the data to be acted on and the MIME type of that data. Different actions are paired with different kinds of data specifications. For example, if the action field isACTION_EDIT, the data field would contain the URI of the document to be displayed for editing. If the action is ACTION_CALL, the data field would be a tel: URI with the number to call. Similarly, if the action is ACTION_VIEW and the data field is an http: URI, the receiving activity would be called upon to download and display whatever data the URI refers to.

When matching an intent to a component that is capable of handling the data, it's often important to know the type of data (its MIME type) in addition to its URI. For example, a component able to display image data should not be called upon to play an audio file.

In many cases, the data type can be inferred from the URI — particularly content: URIs, which indicate that the data is located on the device and controlled by a content provider (see the separate discussion on content providers). But the type can also be explicitly set in the Intent object. The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().

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 object. As it does for actions, the Intent class defines several category constants, including these:

Constant	Meaning
CATEGORY_BROWSABLE	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message.
CATEGORY_GADGET	The activity can be embedded inside of another activity that hosts gadgets.
CATEGORY_HOME	The activity displays the home screen, the first screen the user sees when the device is turned on or when the *Home* button is pressed.
CATEGORY_LAUNCHER	The activity can be the initial activity of a task and is listed in the top-level application launcher.
CATEGORY_PREFERENCE	The target activity is a preference panel.

See the Intent class description for the full list of categories.

The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object.

Extras

Key-value pairs for additional information that should be delivered to the component handling the intent. Just as some actions are paired with particular kinds of data URIs, some are paired with particular extras. For example, an ACTION_TIMEZONE_CHANGED intent has a "time-zone" extra that identifies the new time zone, and ACTION_HEADSET_PLUG has a "state" extra indicating whether the headset is now plugged in or unplugged, as well as a "name" extra for the type of headset. If you were to invent a SHOW_COLOR action, the color value would be set in an extra key-value pair.

The Intent object has a series of put...() methods for inserting various types of extra data and a similar set of get...() methods for reading the data. These methods parallel those for Bundle objects. In fact, the extras can be installed and read as a Bundle using the putExtras() and getExtras() methods.

Flags

Flags of various sorts. Many instruct the Android system how to launch an activity (for example, which task 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). All these flags are defined in the Intent class.

The Android system and the applications that come with the platform employ Intent objects both to send out system-originated broadcasts and to activate system-defined components. To see how to structure an intent to activate a system component, consult the list of intents in the reference.

Intent Resolution

Intents can be divided into two groups:

- *Explicit intents* designate the target component by its name (the component name field, mentioned earlier, has a value set). Since component names would generally not be known to developers of other applications, explicit intents are typically used for application-internal messages — such as an activity starting a subordinate service or launching a sister activity.
- *Implicit intents* do not name a target (the field for the component name is blank). Implicit intents are often used to activate components in other applications.

Android delivers an explicit intent to an instance of the designated target class. Nothing in the Intent object other than the component name matters for determining which component should get the intent.

A different strategy is needed for implicit intents. In the absence of a designated target, the Android system must find the best component (or components) to handle the intent — a single activity or service to perform the requested action or the set of broadcast receivers to respond to the broadcast announcement. It does so by comparing the contents of the Intent object to *intent filters*, structures associated with components that can potentially receive intents. Filters advertise the capabilities of a component and delimit the intents it can handle. They open the component to the possibility of receiving implicit intents of the advertised type. If a component does not have any intent filters, it can receive only explicit intents. A component with filters can receive both explicit and implicit intents.

Only three aspects of an Intent object are consulted when the object is tested against an intent filter:

action
data (both URI and data type)
category

The extras and flags play no part in resolving which component receives an intent.

Intent filters

To inform the system which implicit intents they can handle, activities, services, and broadcast receivers can have one or more intent filters. Each filter describes a capability of the component, a set of intents that the component is willing to receive. It, in effect, filters in intents of a desired type, while filtering out unwanted intents — but only unwanted implicit intents (those that don't name a target class). An explicit intent is always delivered to its target, no matter what it contains; the filter is not consulted. But an implicit intent is delivered to a component only if it can pass through one of the component's filters.