Sending the User to Another App

THIS LESSON TEACHES YOU TO

YOU SHOULD ALSO READ

- Sharing Simple Data

One of Android's most important features is an app's ability to send the user to another app based on an "action" it would like to perform. For example, if your app has the address of a business that you'd like to show on a map, you don't have to build an activity in your app that shows a map. Instead, you can create a request to view the address using an `Intent (/reference/android/content/Intent.html)`. The Android system then starts an app that's able to show the address on a map.

As explained in the first class, Building Your First App (/training/basics/firstapp/index.html), you must use intents to navigate between activities in your own app. You generally do so with an *explicit intent*, which defines the exact class name of the component you want to start. However, when you want to have a separate app perform an action, such as "view a map," you must use an *implicit intent*.

This lesson shows you how to create an implicit intent for a particular action, and how to use it to start an activity that performs the action in another app.

Build an Implicit Intent

Implicit intents do not declare the class name of the component to start, but instead declare an action to perform. The action specifies the thing you want to do, such as *view*, *edit*, *send*, or *get* something. Intents often also include data associated with the action, such as the address you want to view, or the email message you want to send. Depending on the intent you want to create, the data might be a `Uri (/reference/android/net/Uri.html)`, one of several other data types, or the intent might not need data at all.

If your data is a `Uri (/reference/android/net/Uri.html)`, there's a simple `Intent() (/reference/android/content/Intent.html#Intent(java.lang.String, android.net.Uri))` constructor you can use define the action and data.

For example, here's how to create an intent to initiate a phone call using the `Uri (/reference/android/net/Uri.html)` data to specify the telephone number:

```
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```

When your app invokes this intent by calling `startActivity() (/reference/android/app/Activity.html#startActivity(android.content.Intent))`, the Phone app initiates a call to the given phone number.

Here are a couple other intents and their action and `Uri (/reference/android/net/Uri.html)` data pairs:

- View a map:

```
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+Califor
// Or map point based on latitude/longitude
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z param is zoom level
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

- View a web page:

```
Uri webpage = Uri.parse("http://www.android.com");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```

Other kinds of implicit intents require "extra" data that provide different data types, such as a string. You can add one or more pieces of extra data using the various `putExtra()` (/reference/android/content/Intent.html#putExtra(java.lang.String, java.lang.String)) methods.

By default, the system determines the appropriate MIME type required by an intent based on the `Uri` (/reference/android/net/Uri.html) data that's included. If you don't include a `Uri` (/reference/android/net/Uri.html) in the intent, you should usually use `setType()` (/reference/android/content/Intent.html#setType(java.lang.String)) to specify the type of data associated with the intent. Setting the MIME type further specifies which kinds of activities should receive the intent.

Here are some more intents that add extra data to specify the desired action:

- Send an email with an attachment:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
// The intent does not have a URI, so declare the "text/plain" MIME type
emailIntent.setType(HTTP.PLAIN_TEXT_TYPE);
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jon@example.com"}); // recipie
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email subject");
emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message text");
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("content://path/to/email/attachmen
// You can also attach multiple items by passing an ArrayList of Uris
```

- Create a calendar event:

```
Intent calendarIntent = new Intent(Intent.ACTION_INSERT, Events.CONTENT_URI);
Calendar beginTime = Calendar.getInstance().set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance().set(2012, 0, 19, 10, 30);
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, beginTime.getTimeInMi
calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME, endTime.getTimeInMillis
calendarIntent.putExtra(Events.TITLE, "Ninja class");
calendarIntent.putExtra(Events.EVENT_LOCATION, "Secret dojo");
```

> **Note:** This intent for a calendar event is supported only with API level 14 and higher.

> **Note:** It's important that you define your `Intent` (/reference/android/content/Intent.html) to be as specific as possible. For example, if you want to display an image using the `ACTION_VIEW` (/reference/android/content/Intent.html#ACTION_VIEW) intent, you should specify a MIME type of `image/*`. This prevents apps that can "view" other types of data (like a map app) from being triggered by the intent.

Verify There is an App to Receive the Intent

Although the Android platform guarantees that certain intents will resolve to one of the built-in apps (such as the Phone, Email, or Calendar app), you should always include a verification step before invoking an intent.

> **Caution:** If you invoke an intent and there is no app available on the device that can handle the intent, your app will crash.

To verify there is an activity available that can respond to the intent, call `queryIntentActivities()` `(/reference/android/content/pm/PackageManager.html#queryIntentActivities(android.content.Intent, int))` to get a list of activities capable of handling your `Intent (/reference/android/content/Intent.html)`. If the returned `List (/reference/java/util/List.html)` is not empty, you can safely use the intent. For example:

```
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(intent, 0);
boolean isIntentSafe = activities.size() > 0;
```

If `isIntentSafe` is `true`, then at least one app will respond to the intent. If it is `false`, then there aren't any apps to handle the intent.

> **Note:** You should perform this check when your activity first starts in case you need to disable the feature that uses the intent before the user attempts to use it. If you know of a specific app that can handle the intent, you can also provide a link for the user to download the app (see how to link to your product on Google Play (/distribute/googleplay/promote/linking.html)).

Start an Activity with the Intent

Once you have created your `Intent` `(/reference/android/content/Intent.html)` and set the extra info, call `startActivity()` `(/reference/android/app/Activity.html#startActivity(android.content.Intent))` to send it to the system. If the system identifies more than one activity that can handle the intent, it displays a dialog for the user to select which app to use, as shown in figure 1. If there is only one activity that handles the intent, the system immediately starts it.

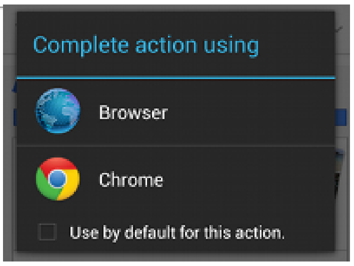

Figure 1. Example of the selection dialog that appears when more than one app can handle an intent.

```
startActivity(intent);
```

Here's a complete example that shows how to create an intent to view a map, verify that an app exists to handle the intent, then start it:

```
// Build the intent
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+Califorr
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Verify it resolves
```

```
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```

Show an App Chooser

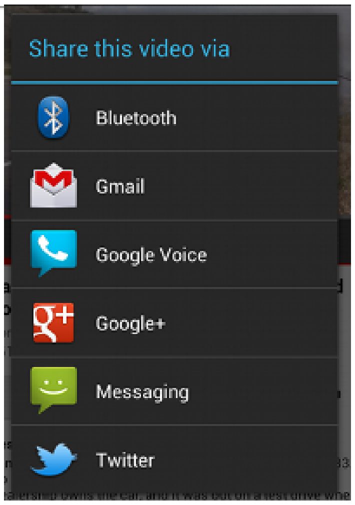

Figure 2. A chooser dialog.

Notice that when you start an activity by passing your Intent (/reference/android/content/Intent.html) to startActivity() (/reference/android/app/Activity.html#startActivity(android.content.Intent)) and there is more than one app that responds to the intent, the user can select which app to use by default (by selecting a checkbox at the bottom of the dialog; see figure 1). This is nice when performing an action for which the user generally wants to use the same app every time, such as when opening a web page (users likely use just one web browser) or taking a photo (users likely prefer one camera).

However, if the action to be performed could be handled by multiple apps and the user might prefer a different app each time—such as a "share" action, for which users might have several apps through which they might share an item—you should explicitly show a chooser dialog as shown in figure 2. The chooser dialog forces the user to select which app to use for the action every time (the user cannot select a default app for the action).

To show the chooser, create an Intent (/reference/android/content/Intent.html) using createChooser() (/reference/android/content/Intent.html#createChooser(android.content.Intent, java.lang.CharSequence)) and pass it to startActivity() (/reference/android/app/Activity.html#startActivity(android.content.Intent)). For example:

```
Intent intent = 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 chooser
Intent chooser = Intent.createChooser(intent, title);
```

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

This displays a dialog with a list of apps that respond to the intent passed to the `createChooser()` `(/reference/android/content/Intent.html#createChooser(android.content.Intent, java.lang.CharSequence))` method and uses the supplied text as the dialog title.