Sending Content to Other Apps

When you construct an intent, you must specify the action you want the intent to "trigger." Android defines several actions, including `ACTION_SEND` (/reference/android/content/Intent.html#ACTION_SEND) which, as you can probably guess, indicates that the intent is sending data from one activity to another, even across process boundaries. To send data to another activity, all you need to do is specify the data and its type, the system will identify compatible receiving activities and display them to the user (if there are multiple options) or immediately start the activity (if there is only one option). Similarly, you can advertise the data types that your activities support receiving from other applications by specifying them in your manifest.

THIS LESSON TEACHES YOU TO

1. Send Text Content
2. Send Binary Content
3. Send Multiple Pieces of Content

YOU SHOULD ALSO READ

- Intents and Intent Filters

Sending and receiving data between applications with intents is most commonly used for social sharing of content. Intents allow users to share information quickly and easily, using their favorite applications.

Note: The best way to add a share action item to an `ActionBar` (/reference/android/app/ActionBar.html) is to use `ShareActionProvider` (/reference/android/widget/ShareActionProvider.html), which became available in API level 14. `ShareActionProvider` (/reference/android/widget/ShareActionProvider.html) is discussed in the lesson about Adding an Easy Share Action (shareaction.html).

Send Text Content

The most straightforward and common use of the `ACTION_SEND` (/reference/android/content/Intent.html#ACTION_SEND) action is sending text content from one activity to another. For example, the built-in Browser app can share the URL of the currently-displayed page as text with any application. This is useful for sharing an article or website with friends via email or social networking. Here is the code to implement this type of sharing:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

If there's an installed application with a filter that matches `ACTION_SEND` (/reference/android/content/Intent.html#ACTION_SEND) and MIME type text/plain, the Android system will run it; if more than one application matches, the system displays a disambiguation dialog (a "chooser") that allows the user to choose an app. If you call `Intent.createChooser()` (/reference/android/content/Intent.html#createChooser(android.content.Intent, java.lang.CharSequence)) for the intent, Android will **always** display the chooser. This has some advantages:

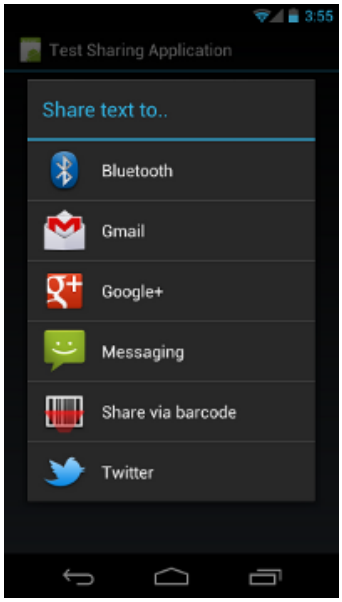

Figure 1. Screenshot of `ACTION_SEND` (/reference/android/content/Intent.html#ACTION_SEND) intent chooser on a handset.

- Even if the user has previously selected a default action for this intent, the chooser will still be displayed.
- If no applications match, Android displays a system message.
- You can specify a title for the chooser dialog.

Here's the updated code:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(Intent.createChooser(sendIntent, getResources().getText(R.string.send_to
```

The resulting dialog is shown in figure 1.

Optionally, you can set some standard extras for the intent: `EXTRA_EMAIL` (/reference/android/content/Intent.html#EXTRA_EMAIL), `EXTRA_CC` (/reference/android/content/Intent.html#EXTRA_CC), `EXTRA_BCC` (/reference/android/content/Intent.html#EXTRA_BCC), `EXTRA_SUBJECT` (/reference/android/content/Intent.html#EXTRA_SUBJECT). However, if the receiving application is not designed to use them, nothing will happen. You can use custom extras as well, but there's no effect unless the receiving application understands them. Typically, you'd use custom extras defined by the receiving application itself.

> **Note:** Some e-mail applications, such as Gmail, expect a `String[]` (/reference/java/lang/String.html) for extras like `EXTRA_EMAIL` (/reference/android/content/Intent.html#EXTRA_EMAIL) and `EXTRA_CC` (/reference/android/content/Intent.html#EXTRA_CC), use `putExtra(String, String[])` (/reference/android/content/Intent.html#putExtra(java.lang.String, java.lang.String[])) to add these to your intent.

Send Binary Content

Binary data is shared using the `ACTION_SEND` (/reference/android/content/Intent.html#ACTION_SEND) action combined with setting the appropriate MIME type and placing the URI to the data in an extra named `EXTRA_STREAM` (/reference/android/content/Intent.html#EXTRA_STREAM). This is commonly used to share an image but can be used to share any type of binary content:

```
Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND);
shareIntent.putExtra(Intent.EXTRA_STREAM, uriToImage);
shareIntent.setType("image/jpeg");
startActivity(Intent.createChooser(shareIntent, getResources().getText(R.string.send_t
```

Note the following:

- You can use a MIME type of `"*/*"`, but this will only match activities that are able to handle generic data streams.
- The receiving application needs permission to access the data the `Uri` points to. There are a number of ways to handle this:
  - Write the data to a file on external/shared storage (such as the SD card), which all apps can read. Use `Uri.fromFile()` to create the `Uri` that can be passed to the share intent. However, keep in mind that not all applications process a `file://` style `Uri`.
  - Write the data to a file in your own application directory using `openFileOutput()` with mode `MODE_WORLD_READABLE` after which `getFileStreamPath()` can be used to return a `File`. As with the previous option, `Uri.fromFile()` will create a `file://` style `Uri` for your share intent.
  - Media files like images, videos and audio can be scanned and added to the system `MediaStore` using `scanFile()`. The `onScanCompleted()` callback returns a `content://` style `Uri` suitable for including in your share intent.
  - Images can be inserted into the system `MediaStore` using `insertImage()` which will return a `content://` style `Uri` suitable for including in a share intent.
  - Store the data in your own `ContentProvider`, make sure that other apps have the correct permission to access your provider (or use per-URI permissions).

Send Multiple Pieces of Content

To share multiple pieces of content, use the ACTION_SEND_MULTIPLE (/reference/android/content/Intent.html#ACTION_SEND_MULTIPLE) action together with a list of URIs pointing to the content. The MIME type varies according to the mix of content you're sharing. For example, if you share 3 JPEG images, the type is still `"image/jpeg"`. For a mixture of image types, it should be `"image/*"` to match an activity that handles any type of image. You should only use `"*/*"` if you're sharing out a wide variety of types. As previously stated, it's up to the receiving application to parse and process your data. Here's an example:

```
ArrayList<Uri> imageUris = new ArrayList<Uri>();
imageUris.add(imageUri1); // Add your image URIs here
imageUris.add(imageUri2);

Intent shareIntent = new Intent();
shareIntent.setAction(Intent.ACTION_SEND_MULTIPLE);
shareIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, imageUris);
shareIntent.setType("image/*");
startActivity(Intent.createChooser(shareIntent, "Share images to.."));
```

As before, make sure the provided URIs (/reference/android/net/Uri.html) point to data that a receiving application can access.