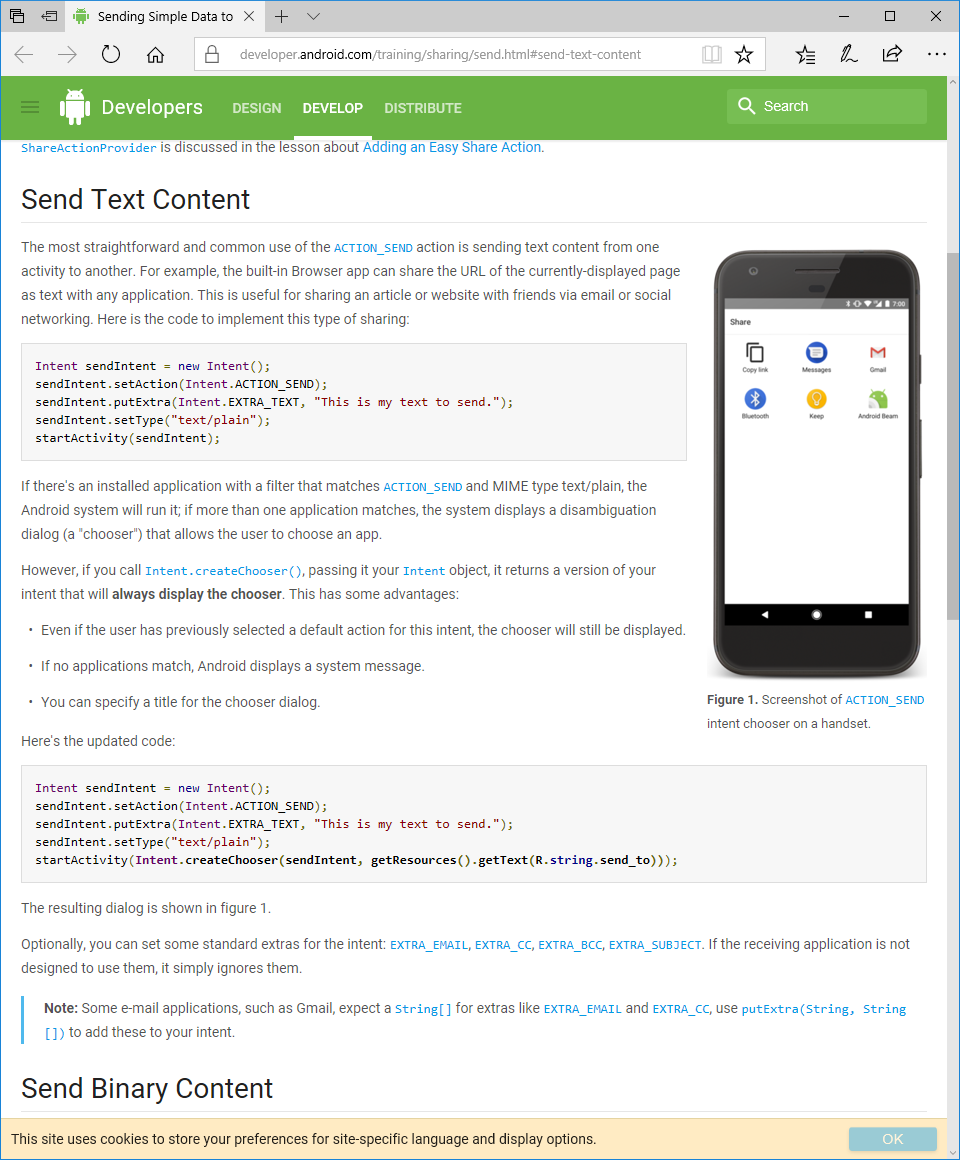
**Data Sharing**

**Sending Text Context**

When you construct an intent, you must specify the action you want the intent to "trigger." Android defines several actions, including 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).

The most straightforward and common use of the 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:

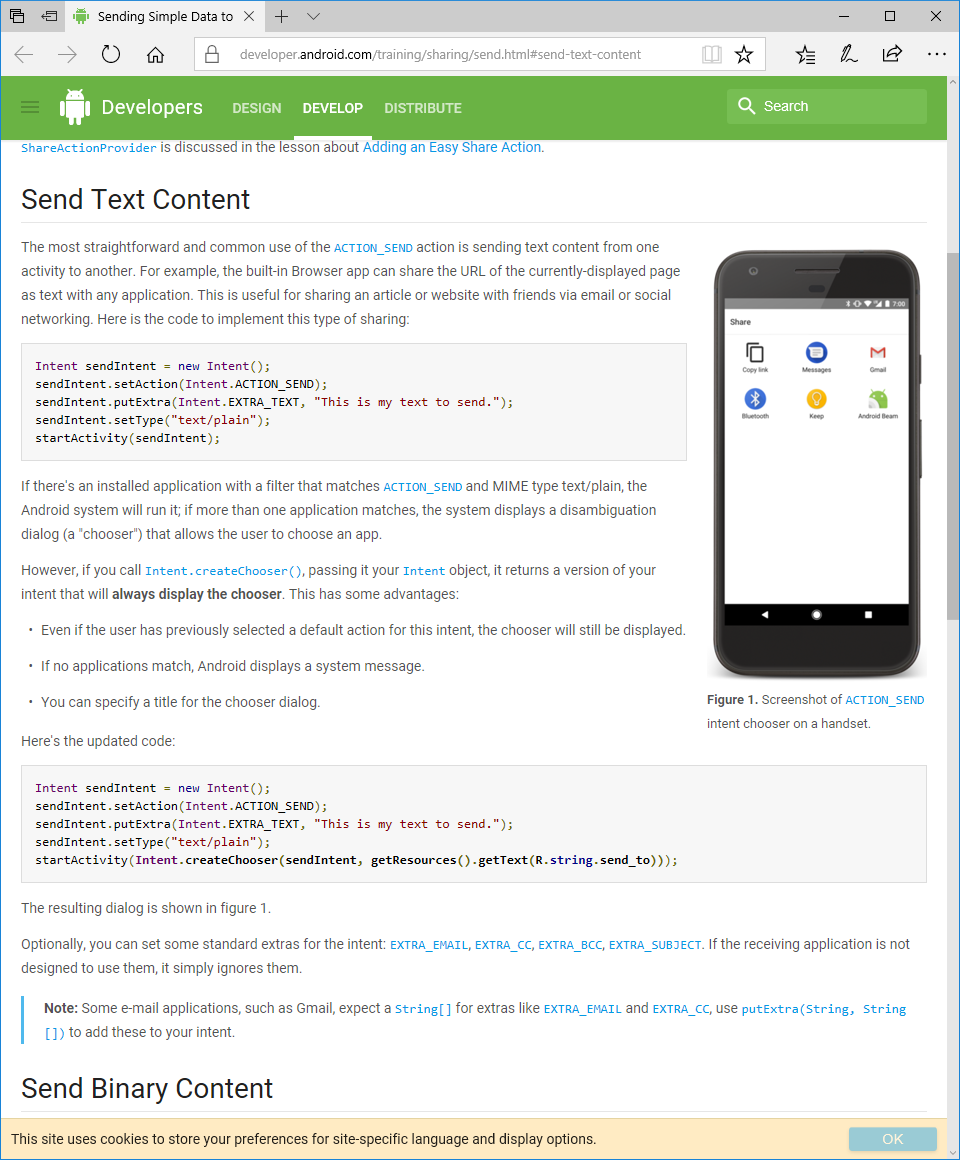


If there is, an installed application with a filter that matches ACTION\_SEND and MIME type text/plain, the Android system will run it; if more than one application matches, the system displays a disambiguation dialog (a "chooser") that allows the user to choose an app.

However, if you call Intent.createChooser(), passing it your Intent object, it returns a version of your intent that will always display the chooser. This has some advantages:

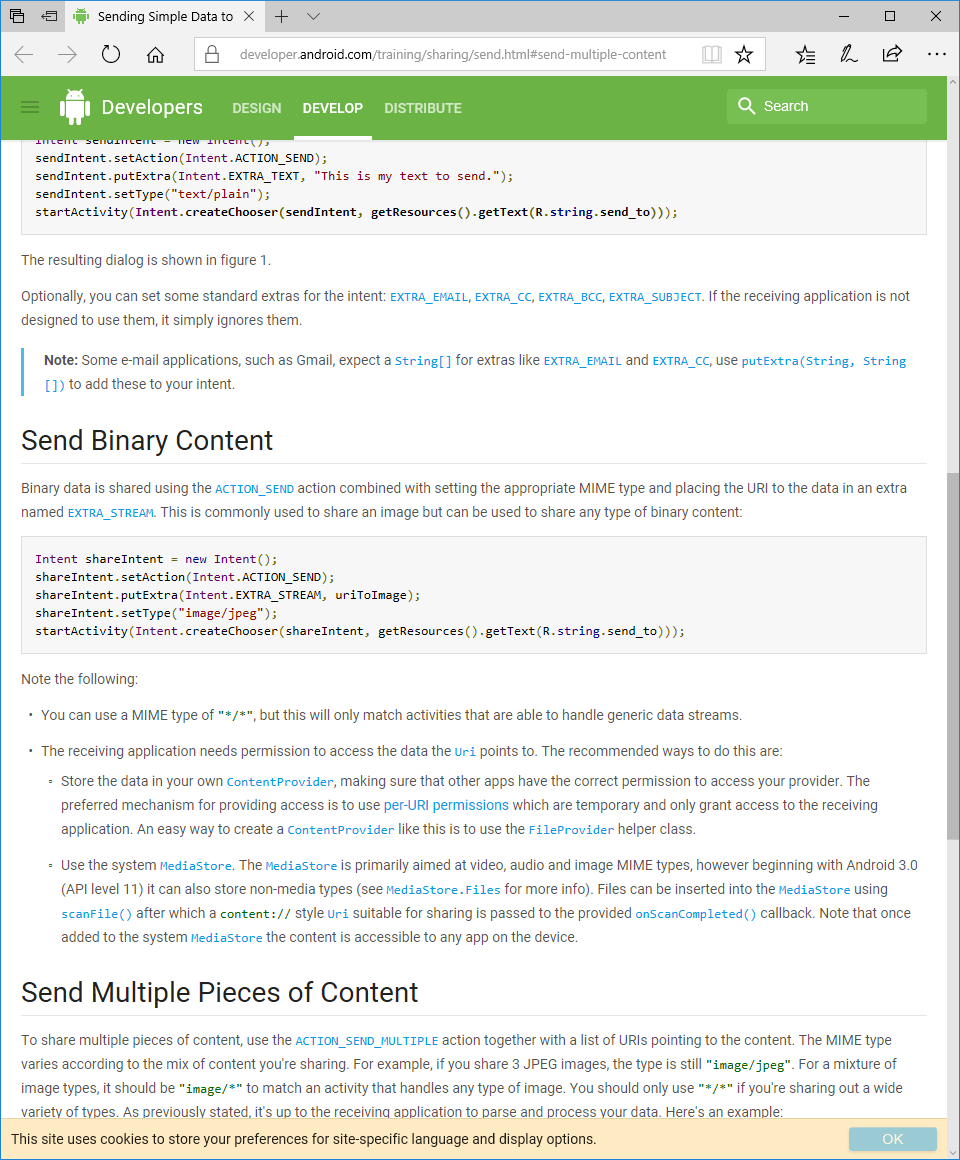
* Even if the user has previously selected a default action for this intent, the chooser will still be displayed.
* If no applications match, Android displays a system message.
* You can specify a title for the chooser dialog.

Here is the updated code:



**Sending Binary Content**

Binary data is shared using the ACTION\_SEND action combined with setting the appropriate MIME type and placing the URI to the data in an extra named EXTRA\_STREAM. This is commonly used to share an image but can be used to share any type of binary content:

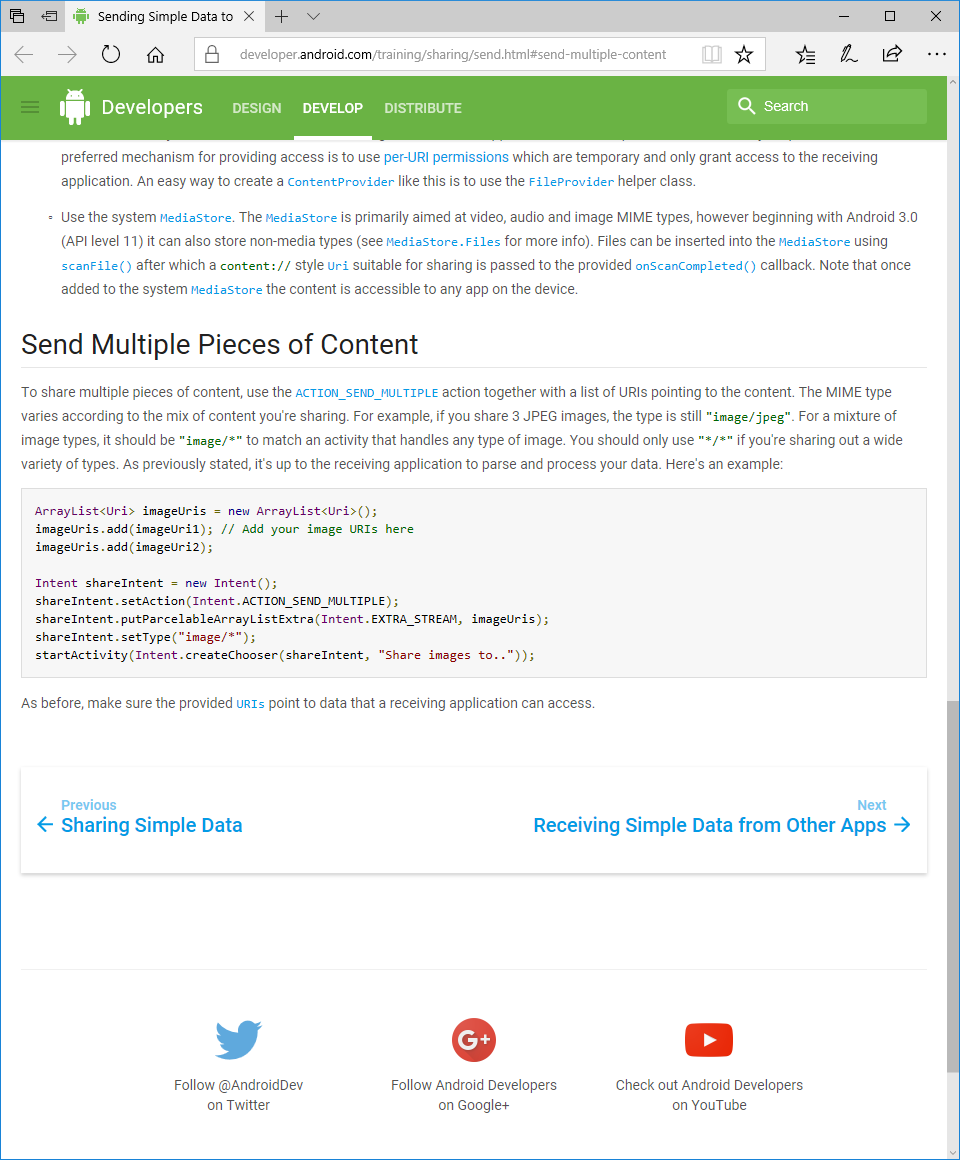


*Note the following:*

* 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](https://developer.android.com/reference/android/net/Uri.html) points to. The recommended ways to do this are:
  + Store the data in your own [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html), making sure that other apps have the correct permission to access your provider. The preferred mechanism for providing access is to use [per-URI permissions](https://developer.android.com/guide/topics/security/permissions.html#uri) which are temporary and only grant access to the receiving application. An easy way to create a [ContentProvider](https://developer.android.com/reference/android/content/ContentProvider.html) like this is to use the [FileProvider](https://developer.android.com/reference/android/support/v4/content/FileProvider.html) helper class.
  + Use the system [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html). The [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html) is primarily aimed at video, audio and image MIME types, however beginning with Android 3.0 (API level 11) it can also store non-media types (see [MediaStore.Files](https://developer.android.com/reference/android/provider/MediaStore.Files.html) for more info). Files can be inserted into the [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html) using [scanFile()](https://developer.android.com/reference/android/media/MediaScannerConnection.html#scanFile(android.content.Context, java.lang.String[], java.lang.String[], android.media.MediaScannerConnection.OnScanCompletedListener)) after which a content:// style [Uri](https://developer.android.com/reference/android/net/Uri.html) suitable for sharing is passed to the provided [onScanCompleted()](https://developer.android.com/reference/android/media/MediaScannerConnection.OnScanCompletedListener.html#onScanCompleted(java.lang.String, android.net.Uri)) callback. Note that once added to the system [MediaStore](https://developer.android.com/reference/android/provider/MediaStore.html) the content is accessible to any app on the device.

**Sending Multiple Pieces of Content**

To share multiple pieces of content, use the 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:

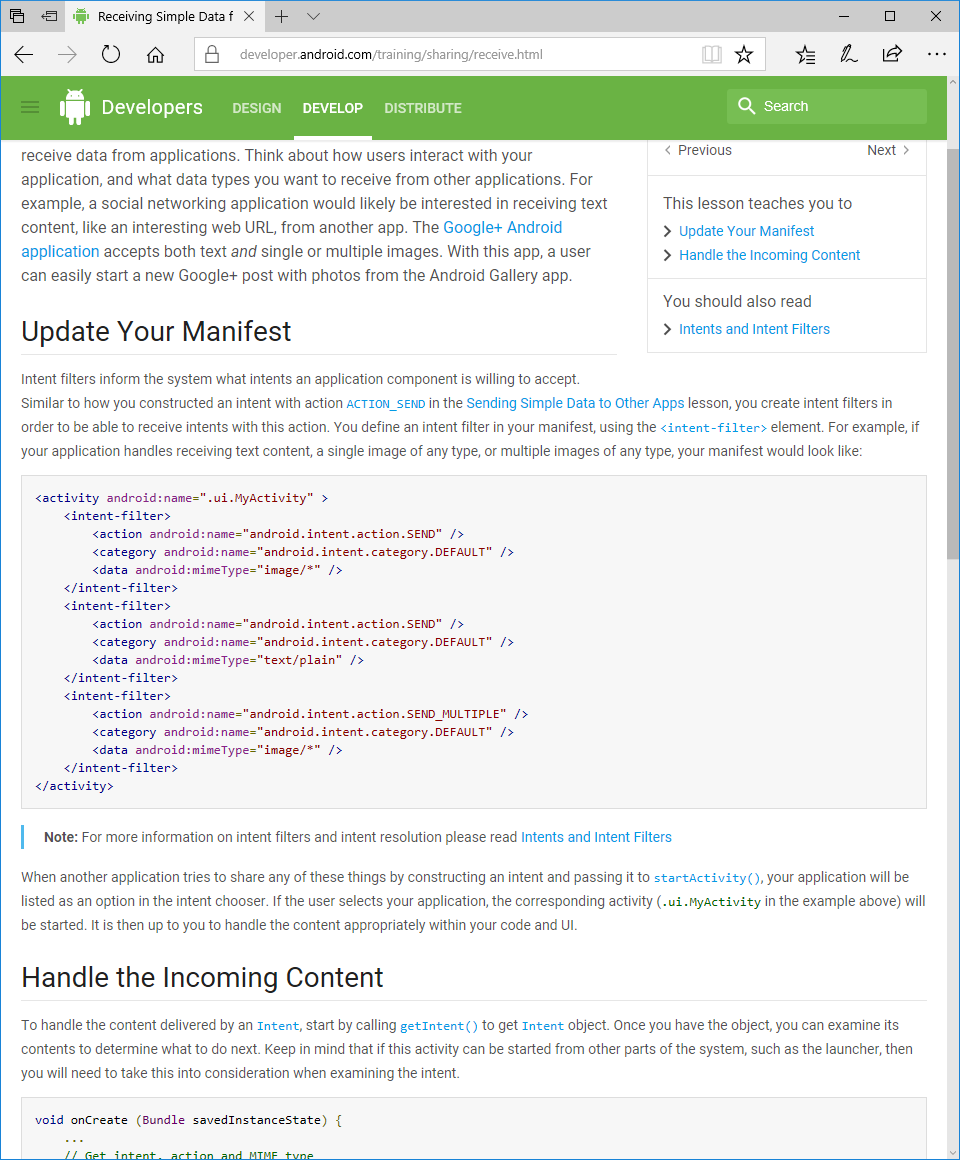


As before, make sure the provided URIs point to data that a receiving application can access.

**Receiving Simple Data from Other Apps**

**Updating AndroidManifest file with intent filter**

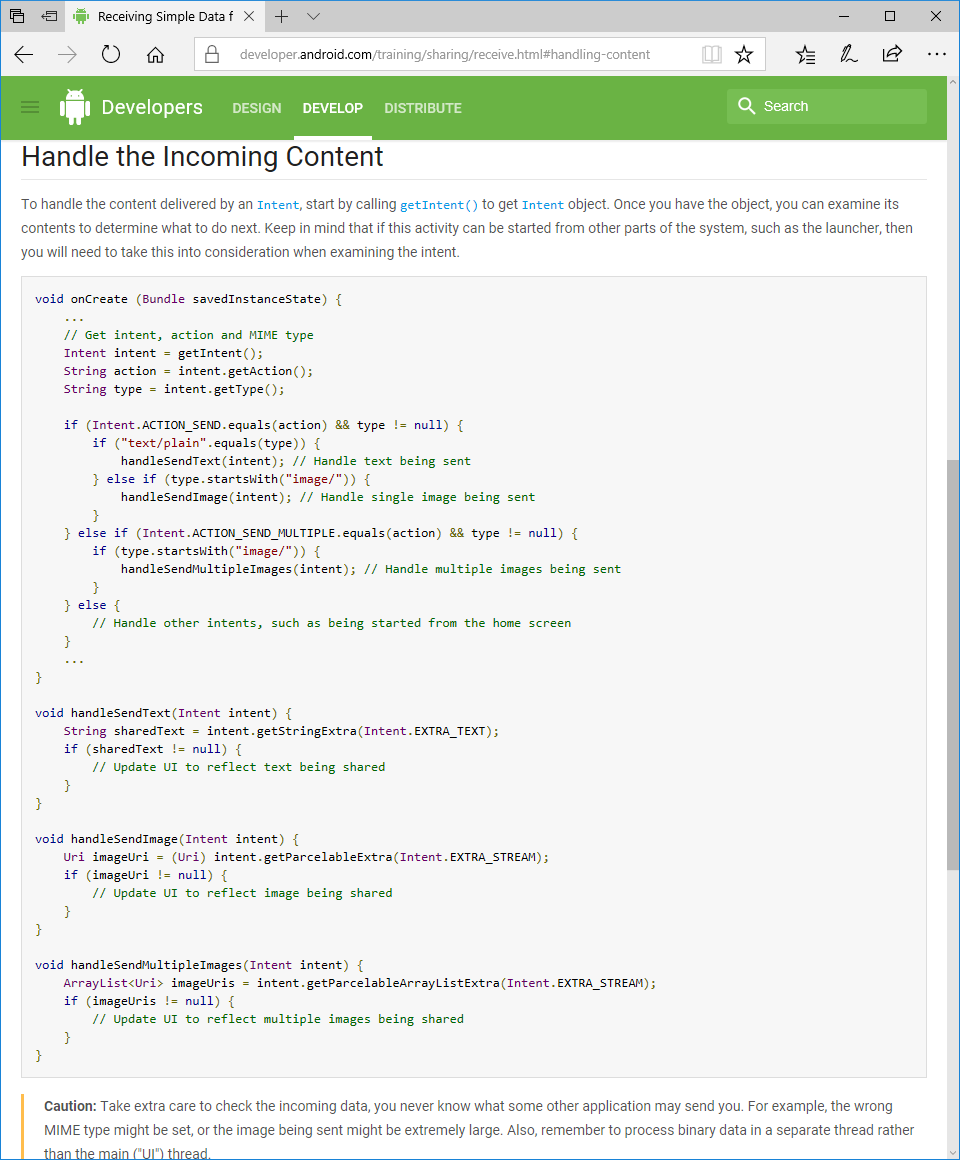
Intent filters inform the system what intents an application component is willing to accept. Similar to how you constructed an intent with action ACTION\_SEND in the Sending Simple Data to Other Apps lesson, you create intent filters in order to be able to receive intents with this action. You define an intent filter in your manifest, using the <intent-filter> element. For example, if your application handles receiving text content, a single image of any type, or multiple images of any type, your manifest would look like:



When another application tries to share any of these things by constructing an intent and passing it to startActivity(), your application will be listed as an option in the intent chooser. If the user selects your application, the corresponding activity (.ui.MyActivity in the example above) will be started. It is then up to you to handle the content appropriately within your code and UI.

**Handle Incoming Content**

To handle the content delivered by an Intent, start by calling getIntent() to get Intent object. Once you have the object, you can examine its contents to determine what to do next. Keep in mind that if this activity can be started from other parts of the system, such as the launcher, then you will need to consider this when examining the intent.

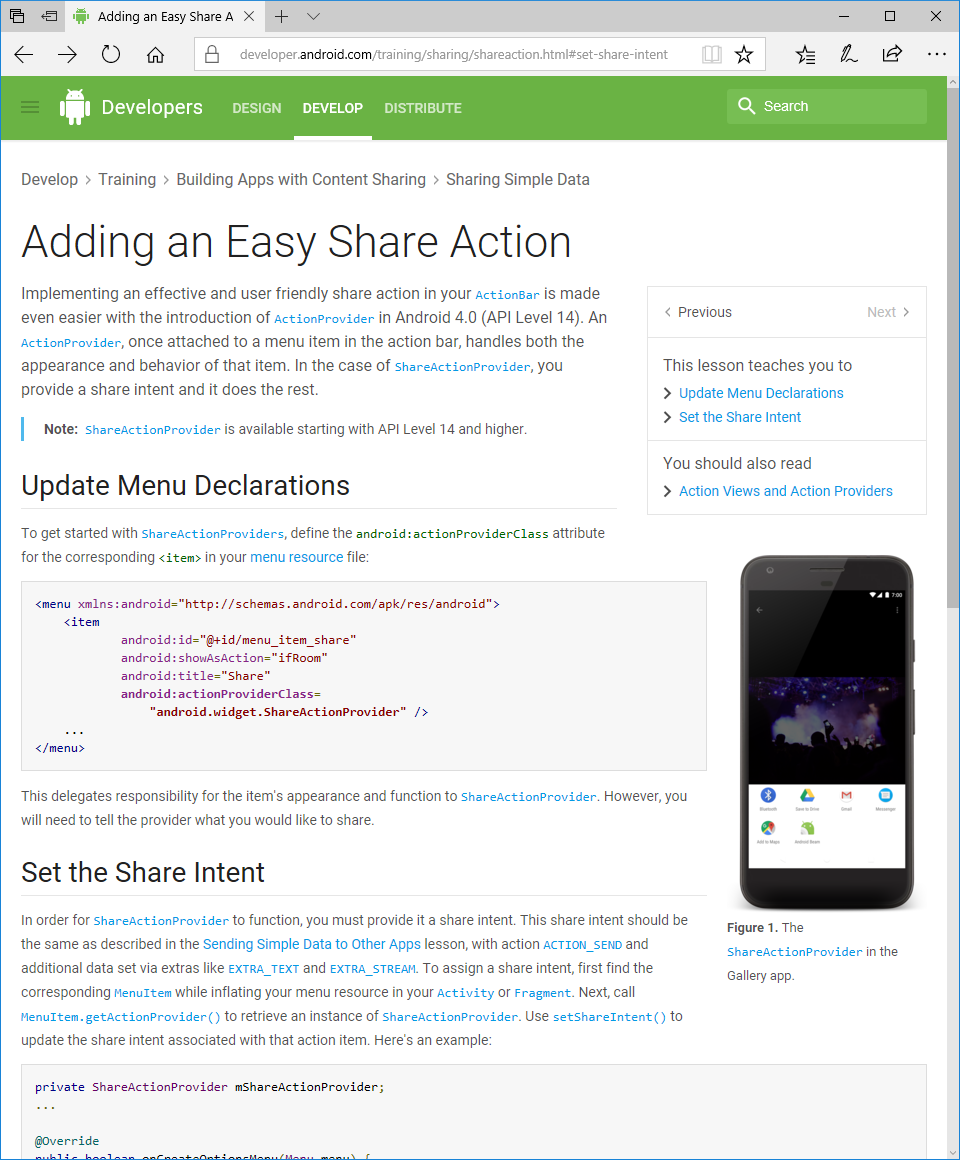


Updating the UI can be as simple as populating an EditText, or it can be more complicated like applying an interesting photo filter to an image. It's really specific to your application what happens next.

**Implementing an Effective and User Friendly Share Action**

**Updating Menu Declarations**

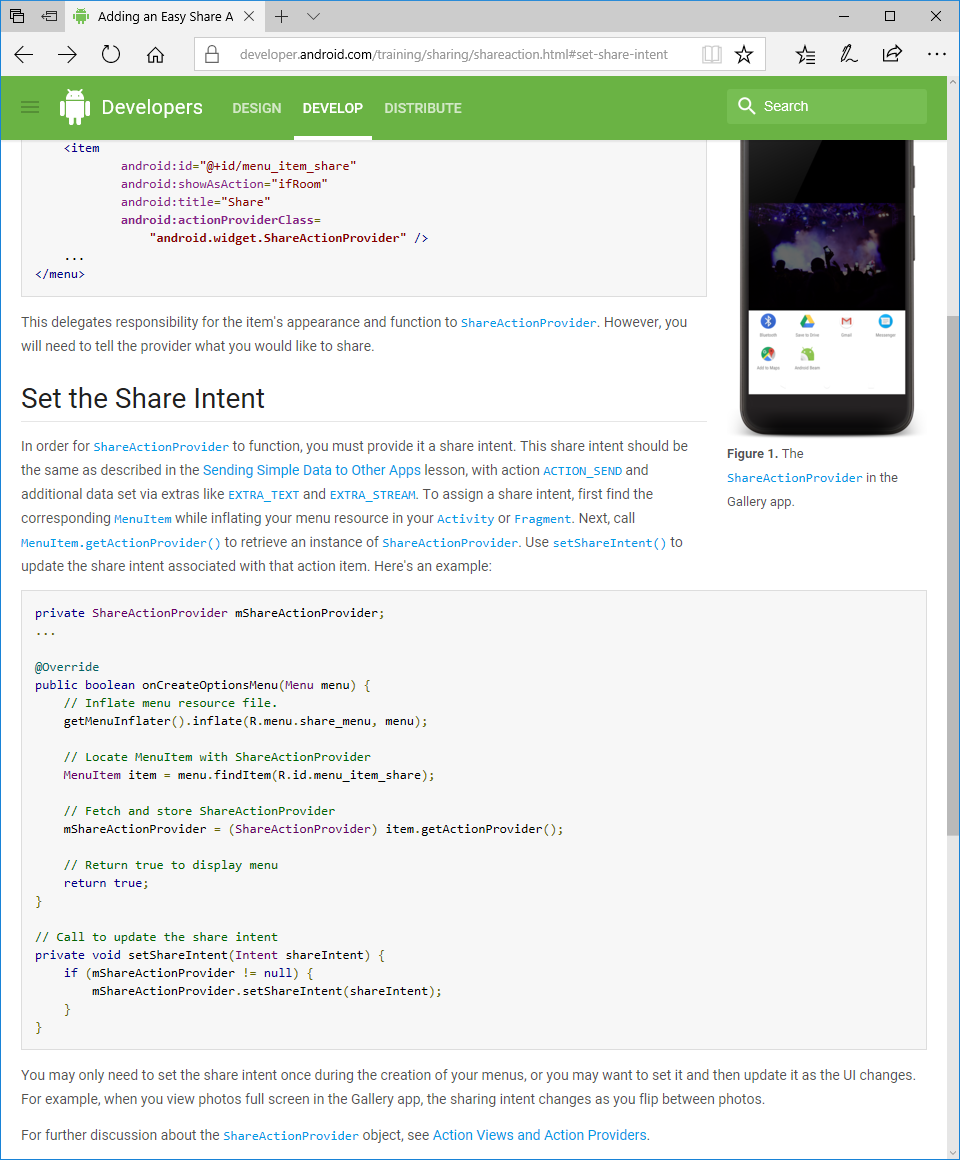
To get started with ShareActionProviders, define the android:actionProviderClass attribute for the corresponding <item> in your menu resource file:



This delegates responsibility for the item's appearance and function to ShareActionProvider. However, you will need to tell the provider what you would like to share.

**Setting the Share Intent**

In order for ShareActionProvider to function, you must provide it a share intent. This share intent should be the same as described in the Sending Simple Data to Other Apps lesson, with action ACTION\_SEND and additional data set via extras like EXTRA\_TEXT and EXTRA\_STREAM. To assign a share intent, first find the corresponding MenuItem while inflating your menu resource in your Activity or Fragment. Next, call MenuItem.getActionProvider() to retrieve an instance of ShareActionProvider. Use setShareIntent() to update the share intent associated with that action item. Here's an example:



You may only need to set the share intent once during the creation of your menus, or you may want to set it and then update it as the UI changes. For example, when you view photos full screen in the Gallery app, the sharing intent changes as you flip between photos.