Using Intent Filters to Service Implicit Intents

If an Intent is a request for an action to be performed on a set of data, how does Android know which application (and component) to use to service the request? Intent Filters are used to register Activities, Services, and Broadcast Receivers as being capable of performing an action on a particular kind of data.

Using Intent Filters, application components tell Android that they can service action requests from others, including components in the same, native, or third-party applications.

To register an application component as an Intent handler, use the intent-filter tag within the component's manifest node.

Using the following tags (and associated attributes) within the Intent Filter node, you can specify a component's supported actions, categories, and data:

❑ action Use the android:name attribute to specify the name of the action being serviced. Actions should be unique strings, so best practice is to use a naming system based on the Java package naming conventions.

❑ category Use the android:category attribute to specify under which circumstances the action should be serviced. Each Intent Filter tag can include multiple category tags. You can specify your own categories or use the standard values provided by Android and listed below:

❑ ALTERNATIVE As you'll see later in this chapter, one of the uses of Intent Filters is to help populate context menus with actions. The alternative category specifi es that this action should be available as an alternative to the default action performed on an item of this data type. For example, where the default action for a contact is to view it, the alternatives could be to edit or delete it.

❑ SELECTED_ALTERNATIVE Similar to the alternative category, but where Alternative will always resolve to a single action using the Intent resolution described below, SELECTED_ALTERNATIVE is used when a list of possibilities is required.

❑ BROWSABLE Specifi es an action available from within the browser. When an Intent is fired from within the browser, it will always specify the browsable category.

❑ DEFAULT Set this to make a component the default action for the data values defi ned by the Intent Filter. This is also necessary for Activities that are launched using an explicit Intent.

❑ GADGET By setting the gadget category, you specify that this Activity can run embedded inside another Activity.

❑ HOME The home Activity is the fi rst Activity displayed when the device starts (the launch screen). By setting an Intent Filter category as home without specifying an action, you are presenting it as an alternative to the native home screen.

❑ LAUNCHER Using this category makes an Activity appear in the application launcher.

❑ data The data tag lets you specify matches for data your component can act on; you can include several schemata if your component is capable of handling more than one. You can use any combination of the following attributes to specify the data that your component supports:

❑ android:host Specifi es a valid host name (e.g., com.google).

❑ android:mimetype Lets you specify the type of data your component is capable of handling. For example, <type android:value="vnd.android.cursor.dir/*"/> would match any Android cursor.

❑ android:path Valid "path" values for the URI (e.g., /transport/boats/)

❑ android:port Valid ports for the specifi ed host

❑ android:scheme Requires a particular scheme (e.g., content or http).

The following code snippet shows how to confi gure an Intent Filter for an Activity that can perform the SHOW_DAMAGE action as either a primary or alternative action. (You'll create earthquake content in the next chapter.)

```
<activity android:name=".EarthquakeDamageViewer"
android:label="View Damage">
<intent-filter>
<action
android:name="com.paad.earthquake.intent.action.SHOW_DAMAGE">
</action>
<category android:name="android.intent.category.DEFAULT"/>
<category
android:name="android.intent.category.ALTERNATIVE_SELECTED"
/>
<data android:mimeType="vnd.earthquake.cursor.item/*"/>
</intent-filter>

</activity>
```

How Android Resolves Intent Filters

The anonymous nature of runtime binding makes it important to understand how Android resolves an implicit Intent into a particular application component.

As you saw previously, when using startActivity, the implicit Intent resolves to a single Activity. If there are multiple Activities capable of performing the given action on the specifi ed data, the "best" of those Activities will be launched.
The process of deciding which Activity to start is called *Intent resolution*. The aim of Intent resolution is to fi nd the best Intent Filter match possible using the following process:

1. Android puts together a list of all the Intent Filters available from the installed packages.

2. Intent Filters that do not match the action or category associated with the Intent being resolved are removed from the list.

   2.1. Action matches are made if the Intent Filter either includes the specifi ed action or has no action specifi ed.

   An Intent Filter will only fail the action match check if it has one or more actions defi ned, where none of them match the action specifi ed by the Intent.

   2.2. Category matching is stricter. Intent Filters must include *all* the categories defi ned in the resolving Intent. An Intent Filter with no categories specifi ed only matches Intents with no categories.

3. Finally, each part of the Intent's data URI is compared to the Intent Filter's data tag. If Intent Filter defi nes the scheme, host/authority, path, or mime type, these values are compared to the Intent's URI. Any mismatches will remove the Intent Filter from the list. Specifying no data values in an Intent Filter will match with all Intent data values.

   3.1. The mime type is the data type of the data being matched. When matching data types, you can use wild cards to match subtypes (e.g., earthquakes/*). If the Intent Filter specifies a data type, it must match the Intent; specifying no data type resolves to all of them.

   3.2. The scheme is the "protocol" part of the URI — for example, http:, mailto:, or tel:.

   3.3. The host name or "data authority" is the section of the URI between the scheme and the path (e.g., www.google.com). For a host name to match, the Intent Filter's scheme must also pass.

   3.4. The data path is what comes after the authority (e.g., /ig). A path can only match if the scheme and host-name parts of the data tag also match.

4. If more than one component is resolved from this process, they are ordered in terms of priority, with an optional tag that can be added to the Intent Filter node. The highest ranking component is then returned.

Native Android application components are part of the Intent resolution process in exactly the same way as third-party applications. They do not have a higher priority and can be completely replaced with new Activities that declare Intent Filters that service the same action requests.

Responding to Intent Filter Matches

When an application component is started through an implicit Intent, it needs to fi nd the action it is to perform and the data upon which to perform it.

Call the getIntent method — usually from within the onCreate method — to extract the Intent used to launch a component, as shown below:

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
Intent intent = getIntent();
}
```

Use the getData and getAction methods to fi nd the data and action of the Intent. Use the type-safe get<type>Extra methods to extract additional information stored in its extras Bundle.

```
String action = intent.getAction();
Uri data = intent.getData();
```

Passing on Responsibility

You can use the startNextMatchingActivity method to pass responsibility for action handling to the next best matching application component, as shown in the snippet below:

```
Intent intent = getIntent();
if (isAfterMidnight)
startNextMatchingActivity(intent);
```

This allows you to add additional conditions to your components that restrict their use beyond the ability of the Intent Filter–based Intent resolution process.

In some cases, your component may wish to perform some processing, or offer the user a choice, before passing the Intent on to the native handler.

Select a Contact Example

In this example, you'll create a new sub-Activity that services the PICK_ACTION for contact data. It displays each of the contacts in the contact database and lets the user select one, before closing and returning its URI to the calling Activity.

It's worth noting that this example is somewhat contrived. Android already supplies an Intent Filter for picking a contact from a list that can be invoked by using the content:/contacts/people/ *URI in an implicit Intent. The purpose of this exercise is to demonstrate the form, even if this particular implementation isn't overly useful.*

1. Create a new ContactPicker project that includes a ContactPicker Activity.

```
package com.paad.contactpicker;
import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.AdapterView.OnItemClickListener;
public class ContactPicker extends Activity {
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
}
}
```

2. Modify the main.xml layout resource to include a single ListView control. This control will be used to display the contacts.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<ListView
android:id="@+id/contactListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
</LinearLayout>
```

3. Create a new listitemlayout.xml layout resource that includes a single Text View. This will be used to display each contact in the List View.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:id="@+id/itemTextView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:padding="10px"
android:textSize="16px"
android:textColor="#FFF"
/>
</LinearLayout>
```

4. Return to the ContactPicker Activity. Override the onCreate method, and extract the data path from the calling Intent.

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
Intent intent = getIntent();
String dataPath = intent.getData().toString();
```

4.1. Create a new data URI for the people stored in the contact list, and bind it to the List View using a SimpleCursorArrayAdapter.

The SimpleCursorArrayAdapter *lets you assign Cursor data, used by Content Providers, to Views. It's used here without further comment but is examined in more detail later in this chapter.*

```
final Uri data = Uri.parse(dataPath + "people/");
final Cursor c = managedQuery(data, null, null, null, null);
String[] from = new String[] {People.NAME};
int[] to = new int[] { R.id.itemTextView };
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
R.layout.listitemlayout,
c,
from,
to);
ListView lv = (ListView)findViewById(R.id.contactListView);
lv.setAdapter(adapter);
```

4.2. Add an ItemClickListener to the List View. Selecting a contact from the list should return a path to the item to the calling Activity.

```
lv.setOnItemClickListener(new OnItemClickListener() {
public void onItemClick(AdapterView<?> parent, View view, int pos,
long id) {
// Move the cursor to the selected item
c.moveToPosition(pos);
// Extract the row id.
int rowId = c.getInt(c.getColumnIndexOrThrow("_id"));
// Construct the result URI.
Uri outURI = Uri.parse(data.toString() + rowId);
Intent outData = new Intent();
outData.setData(outURI);
setResult(Activity.RESULT_OK, outData);
finish();
```

```
}
});
```

4.3. Close off the onCreate method.

```
}
```

5. Modify the application manifest and replace the intent-filter tag of the Activity to add support for the pick action on contact data.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.paad.contactpicker">
<application android:icon="@drawable/icon">
<activity android:name="ContactPicker"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.PICK"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:path="contacts"
android:scheme="content">
</data>
</intent-filter>
</activity>
</application>
</manifest>
```

6. This completes the sub-Activity. To test it, create a new test harness ContentPickerTester Activity. Create a new layout resource — contentpickertester — that includes a TextView to display the selected contact and a Button to start the sub-Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:id="@+id/selected_contact_textview"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
<Button
android:id="@+id/pick_contact_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Pick Contact"
/>
</LinearLayout>
```

7. Override the onCreate method of the ContentPickerTester to add a Click Listener to the button so that it implicitly starts a new sub-Activity by specifying the PICK_ACTION and the contact database URI (content://contacts/).

```
package com.paad.contactpicker;
import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class ContentPickerTester extends Activity {
public static final int PICK_CONTACT = 1;
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.contentpickertester);
Button button = (Button)findViewById(R.id.pick_contact_button);
```

```
button.setOnClickListener(new OnClickListener() {
public void onClick(View _view) {
Intent intent = new Intent(Intent.ACTION_PICK,
Uri.parse("content://contacts/"));
startActivityForResult(intent, PICK_CONTACT);
}
});
}
}
```

8. When the sub-Activity returns, use the result to populate the Text View with the selected contact's name.

```
@Override
public void onActivityResult(int reqCode, int resCode, Intent data) {
super.onActivityResult(reqCode, resCode, data);
switch(reqCode) {
case (PICK_CONTACT) : {
if (resCode == Activity.RESULT_OK) {
Uri contactData = data.getData();
Cursor c = managedQuery(contactData, null, null, null, null);
c.moveToFirst();
String name;
name = c.getString(c.getColumnIndexOrThrow(People.NAME));
TextView tv;
tv = (TextView)findViewById(R.id.selected_contact_textview);
tv.setText(name);
}
break;
}
}
}
```

9. With your test harness complete, simply add it to your application manifest. You'll also need to add a READ_CONTACTS permission within a uses-permission tag, to allow the application to access the contacts database.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.paad.contactpicker">
<application android:icon="@drawable/icon">
<activity android:name=".ContactPicker"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.PICK"/>
<category android:name="android.intent.category.DEFAULT"/>
<data android:path="contacts" android:scheme="content"/>
</intent-filter>
</activity>
<activity android:name=".ContentPickerTester"
android:label="Contact Picker Test">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

When your Activity is running, press the button. The contact picker Activity should be shown as in Figure 5-1.

Figure 5-1

Once you select a contact, the parent Activity should return to the foreground with the selected contact name displayed, as shown in Figure 5-2.

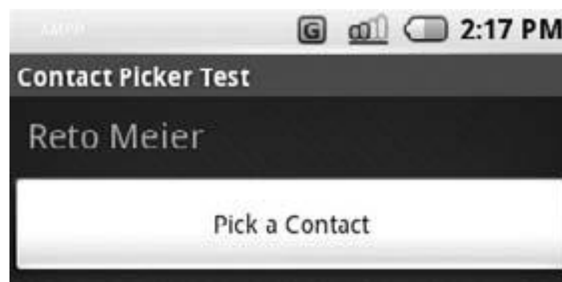

Figure 5-2