Chapter 4:
Intents and Notification

Outline

- Understanding the concept of intents
- Using the Intent object to link activities
- Intent filters to selectively connect to other activities
- Displaying alerts using notifications, toast and dialogs

Introduction

- **Intents** are **asynchronous messaging objects** which allow the application to request functionality from other components of the Android system, e.g. from other *Services* or *Activities*. Abstract description of an *operation* to be performed or *function* that one activity requires another activity to perform.
- **Intents** allow to combine loosely coupled components to perform certain tasks.
- For example the application could implement sharing of data via an Intent and all components which allow sharing of data would be available for the user to select.
- A facility for performing late runtime binding between the code in different applications (“glue” between activities)
- Applications register themselves to an *Intent* via an **IntentFilter**. When an app dispatches an intent, it’s possible that several different activities might be registered (with <intent-filter>) to provide the desired operation.

3 fundamental use cases

- To start an **activity**
- To start a **service**
- To deliver a **broadcast**

To start an activity

- You can start a new instance of an Activity by passing an Intent to **startActivity()**. The Intent object describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call **startActivityForResult()**. Your activity receives the result as a separate Intent object in your activity‘s **onActivityResult()** callback.

To start a service

- You can start a service to perform a one-time operation (such as download a file) by passing an Intent to **startService()**.
  The Intent describes the service to start and carries any necessary data.
- If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to **bindService()**.

Caution

- To ensure an app is secure, always use an **explicit intent** when starting a Service and do not declare intent filters for services.
- Using an implicit intent to start a service is a security hazard because you cannot be certain what service will respond to the intent, and the user cannot see which service starts.

To deliver a broadcast

- The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.
- You can deliver a broadcast to other apps by passing
  an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

Intent Resolution - Intent Types

- **Explicit intents** specify the component (activity, service or broadcast) to start by name (the fully-qualified class name).
  - You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.
  - For example, start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a *general action* to perform, which allows a component from another app to handle it.
  - For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

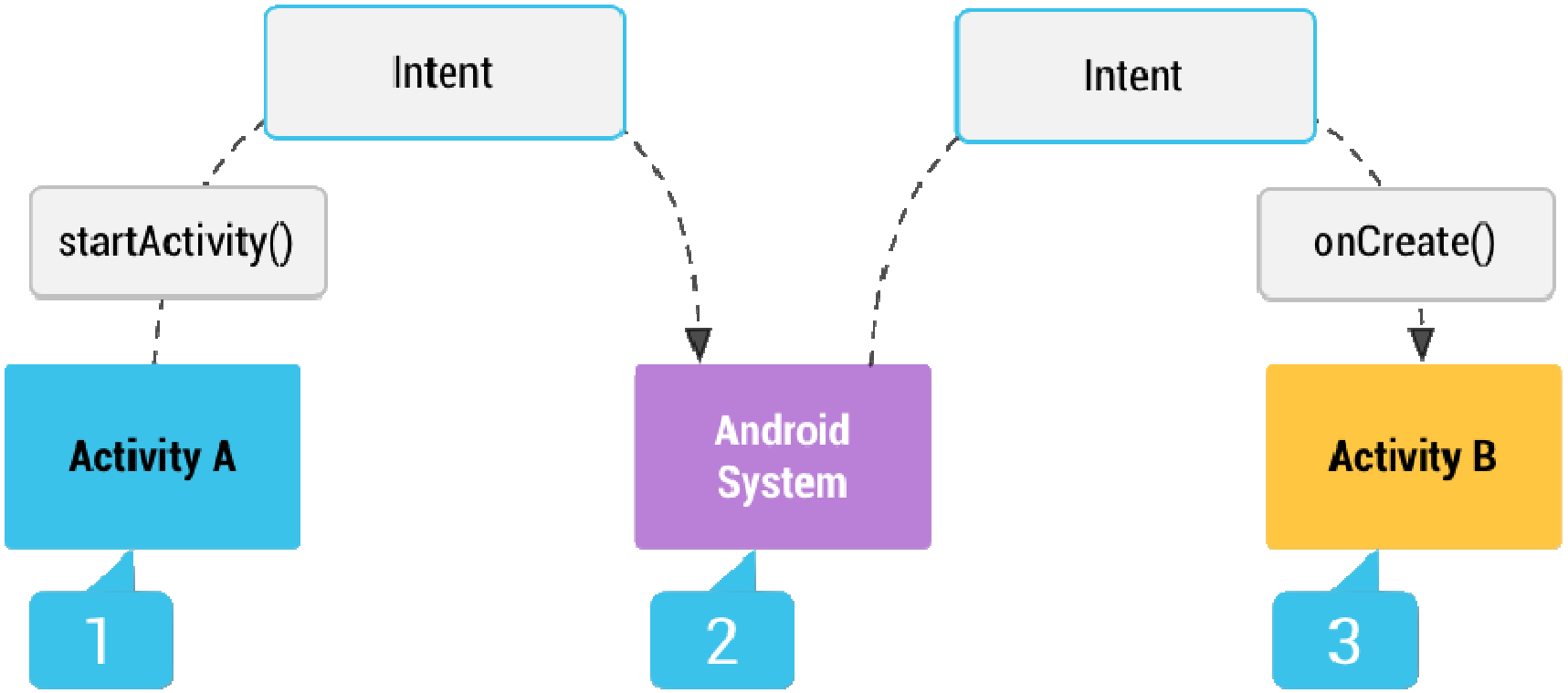


Figure 1. Illustration of how an implicit intent is delivered through the system to start another activity:
[1] Activity A creates an Intent with an action description and passes it to startActivity().
[2] The Android System searches all apps for an intent filter that matches the intent. When a match is found, [3] the system starts the matching activity (Activity B) by invoking its onCreate() method and passing it the Intent.

Starts an activity with Intent

- When you create an **explicit intent** to start an activity or service, the **system immediately starts** the app component specified in the Intent object.
- When you create an **implicit intent**, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

Intent filter

- An **intent filter** is an expression in an app's manifest file that specifies the type of intents that the component would like to receive (and service).
- For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

Intent object

- The primary information contained in an Intent is the following:
  - **Component name -** The name of the component to start (optional). Without a component name, the intent is **implicit** and the system decides which component should receive the intent based on the other intent information

```
public void onClick(View view) {
    startActivity(new Intent("net.learn2develop.SecondActivity"));
}
```

  - **Action -** The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
  - **Data**: The data to operate on, such as a person record in the contacts database, expressed as a Uri.

Action

- Some examples of action/data pairs are:
- **ACTION_VIEW** ***content://contacts/people/1*** -- Display information about the person whose identifier is "1".
- **ACTION_DIAL** ***content://contacts/people/1*** -- Display the phone dialer with the person filled in.
- **ACTION_VIEW** ***tel:123*** -- Display the phone dialer with the given number filled in. Note how the VIEW action does what what is considered the most reasonable thing for a particular URI.
- **ACTION_DIAL** ***tel:123*** -- Display the phone dialer with the given number filled in.
- **ACTION_EDIT** ***content://contacts/people/1*** -- Edit information about the person whose identifier is "1".
- **ACTION_VIEW** ***content://contacts/people/*** -- Display a list of people, which the user can browse through. This example is a typical top-level entry into the Contacts application, showing you the list of people. Selecting a particular person to view would result in a new intent { **ACTION_VIEW** ***content://contacts/N*** } being used to start an activity to display that person.

Secondary attributes

- **category** -- Gives additional information about the action to execute.
  - For example, CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application, while CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.
- **type** -- Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.
- **extras** -- This is a Bundle of any additional information. This can be used to provide extended information to the component. For example, if we have a action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

Building an Explicit Intent

- For example, if you built a service in your app, named DownloadService, designed to download a file from the web, you can start it with the following code:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

- The Intent(Context, Class) constructor supplies the app Context and the component a Class object (DownloadService). As such, this intent explicitly starts the DownloadService class in the app.

Implicit Intent: Sending the User to Another App

- Send the user to another app based on an "action" it would like to perform
- Scenario:
  - 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. The Android system then starts an app that's able to show the address on a map.

Implicit Intent

- Using an implicit intent is useful when your app cannot perform the action, but other apps probably can and you'd like the user to pick which app to use.
- For example, if you have content you want the user to share with other people, create an intent with the ACTION_SEND action and add extras that specify the content to share. When you call startActivity() with that intent, the user can pick an app through which to share the content.

Examples

- Initiate a phone call

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

- View a Web page

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

- View a map

```
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+California");
// 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);
```

- add one or more pieces of extra data using the various putExtra() methods

Verify There is an App to Receive the Intent

- If you invoke an intent and there is no app available on the device that can handle the intent, your app will crash.
- Perform this check when activity first starts:

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

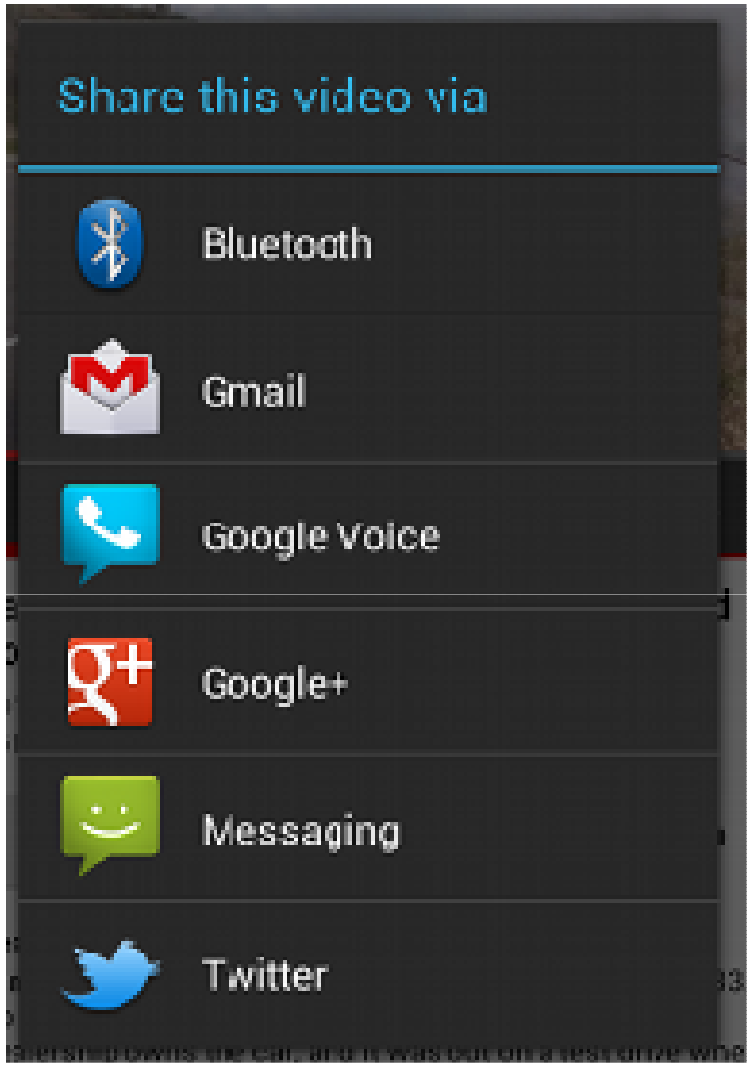
- If isIntentSafe is true, then at least one app will respond to the intent.
- If it is false, then there aren't any apps to handle the intent.

Building an Implicit Intent

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

- In this case, a URI is not used, but the intent's data type is declared to specify the content carried by the extras.

- When startActivity() is called, the system examines all of the installed apps to determine which ones can handle this kind of intent (an intent with the ACTION_SEND action and that carries "text/plain" data).
- If there's only one app that can handle it, that app opens immediately and is given the intent. If multiple activities accept the intent, the system displays a dialog so the user can pick which app to use.



- To show the chooser, create an Intent using createChooser() and pass it to startActivity().
- This displays a dialog with a list of apps that respond to the intent passed to the **createChooser()** method and uses the supplied text as the dialog title.

```
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);

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

Explicit vs Implicit

- **Explicit intent**: declare the class name of the component to start
- **Implicit intent**:
  - declare an **action** to perform, such as *view*, *edit*, *send*, or *get* something.
  - May include **data** associated with the action, such as the address you want to view, or the email message you want to send. Can be Uri, other data types, or no data

Receiving an Intent <intent-filter>

- Declare one or more intent filters for each of your app components with an **<intent-filter>** element in **AndroidManifest** file.
- <intent-filter> element defines how an activity can be invoked by another activity
- Each intent filter specifies the **type of intents** it accepts based on the intent's **action**, **data**, and **category**.
- The system will deliver an implicit intent to your app component only if the intent can pass through one of your intent filters
- An app component should **declare separate filters for each unique job** it can do. For example, one activity in an image gallery app may have two filters: one filter to view an image, and another filter to edit an image.

<intent-filter>

- Each intent filter is defined by an <intent-filter> element in the app's manifest file, nested in the corresponding app component (such as an <activity> element).
- <action>
  - Declares the intent action accepted, in the name attribute. The value must be the literal string value of an action, not the class constant.
- <data>
  - Declares the type of data accepted, using one or more attributes that specify various aspects of the data URI (scheme, host, port, path, etc.) and MIME type.
- <category>
  - Declares the intent category accepted, in the name attribute. The value must be the literal string value of an action, not the class constant. In order to receive implicit intents, you **must include** the CATEGORY_DEFAULT category in the intent filter.

Example

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

Walkthrough:
Linking Activities using **Intent**

- Import **android.content.Intent**
- Can be used with **startActivity** to launch an Activity
- In *AndroidManifest.xml*: **<intent-filter>** element defines how your activity can be invoked by another activity
- In *MainActivity.java*:
  - **startActivity()** method invokes another activity but does not return a result to the current activity.
  - **startActivityForResult()** - start an activity that returns a result to the current activity

Returning results from an intent

- startActivityForResult() - start an activity that returns a result to the current activity

```
startActivityForResult(new Intent(
    "net.learn2develop.SecondActivity"),
    request_Code);
```

- request code: an integer value that identifies an activity you are calling. You need this request code to determine which activity is actually returned

- On SecondActivity, use an Intent object to send data back via the setData() method. The setResult() method sets a result code (either RESULT_OK or RESULT_CANCELLED) and the data (an Intent object) to be returned back to the calling activity.
- The finish() method closes the activity and returns control back to the calling activity.

```
Intent data = new Intent();

//---get the EditText view---
EditText txt_username =
    (EditText) findViewById(R.id.txt_username);

//---set the data to pass back---
data.setData(Uri.parse(
    txt_username.getText().toString()));
setResult(RESULT_OK, data);

//---closes the activity---
finish();
```

- In the calling activity, you need to implement the **onActivityResult()** method, which is called whenever an activity returns:

```
public void onActivityResult(int requestCode, int resultCode,
Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this,data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

Passing data using an intent object

- Passing data from Activity 1 to 2

```
Intent i = new
        Intent("net.learn2develop.PassingDataSecondActivity");

//---use putExtra() to add new key/value pairs---
i.putExtra("str1", "This is a string");
i.putExtra("age1", 25);

//---use a Bundle object to add new key/values
// pairs---
Bundle extras = new Bundle();
extras.putString("str2", "This is another string");
extras.putInt("age2", 35);

//---attach the Bundle object to the Intent object---
i.putExtras(extras);

//---start the activity to get a result back---
startActivityForResult(i, 1);
```

Passing data using an intent object

- Receiving data on Activity 2 from Activity 1

```
//---get the data passed in using getStringExtra()---
Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();

//---get the data passed in using getIntExtra()---
Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();

//---get the Bundle object passed in---
Bundle bundle = getIntent().getExtras();

//---get the data using the getString()---
Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();

//---get the data using the getInt() method---
Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();
```

Passing data using an intent object

- Passing data from Activity 2 to 1

```
//---use an Intent object to return data---
Intent i = new Intent();

//---use the putExtra() method to return some
// value---
i.putExtra("age3", 45);

//---use the setData() method to return some value---
i.setData(Uri.parse(
        "Something passed back to main activity"));

//---set the result with OK and the Intent object---
setResult(RESULT_OK, i);

//---destroy the current activity---
finish();
```

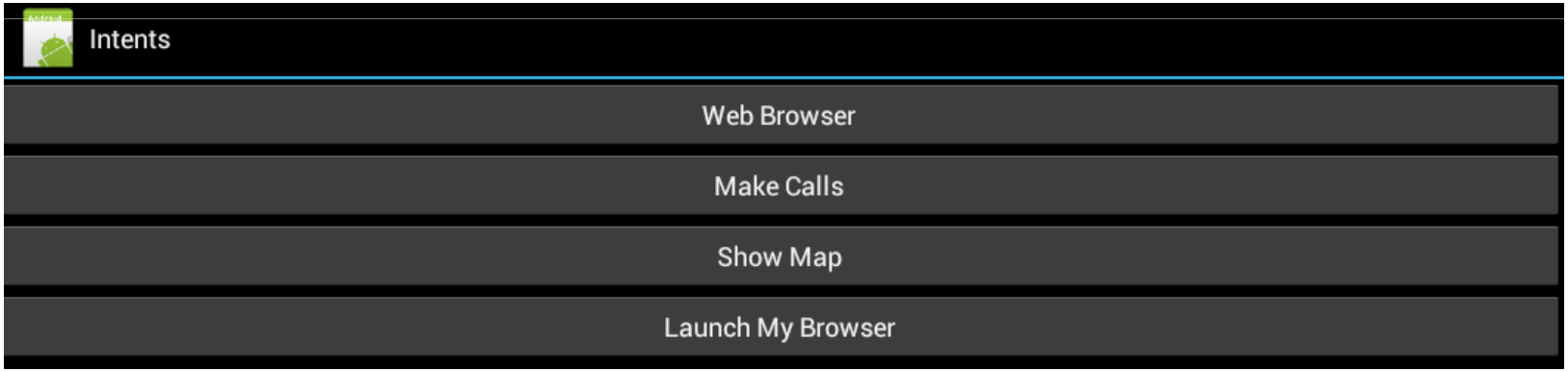
Passing data using an intent object

- Receiving data on Activity 1 from Activity 2

```
public void onActivityResult(int requestCode,
int resultCode, Intent data)
{
    //---check if the request code is 1---
    if (requestCode == 1) {

        //---if the result is OK---
        if (resultCode == RESULT_OK) {

            //---get the result using getIntExtra()---
            Toast.makeText(this, Integer.toString(
                data.getIntExtra("age3", 0)),
                Toast.LENGTH_SHORT).show();

            //---get the result using getData()---
            Toast.makeText(this, data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

Example: Calling Built-in Applications using Intent

- UI with buttons to invoke built-in applications in Android (such as Maps, Phone, Contacts, and Browser).



UI with buttons to invoke actions

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_webbrowser"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Web Browser"
    android:onClick="onClickWebBrowser" />

<Button
    android:id="@+id/btn_makecalls"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Make Calls"
    android:onClick="onClickMakeCalls"  />

<Button
    android:id="@+id/btn_showMap"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Show Map"
    android:onClick="onClickShowMap" />

</LinearLayout>
```

Intent

- *action* and *data.*
  - *action* describes what is to be performed, such as editing an item, viewing the content of an item, and so on.
    - ACTION_VIEW
    - ACTION_DIAL
    - ACTION_PICK
  - *data specifies what is affected,* such as a person in the Contacts database. The data is specified as an Uri object.
    - www.google.com
    - tel:+651234567
    - geo:37.827500,-122.481670
    - content://contacts

Callback methods to invoke actions

```
public class IntentsActivity extends Activity {

    int request_Code = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickWebBrowser(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                        Uri.parse("http://www.amazon.com"));
        startActivity(i);
    }

    public void onClickMakeCalls(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_DIAL,
                        Uri.parse("tel:+651234567"));
        startActivity(i);
    }

    public void onClickShowMap(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                        Uri.parse("geo:37.827500,-122.481670"));
        startActivity(i);
    }
}
```

Using Intent Filters

- When using an Intent object with categories, all categories added to the Intent object must fully match those defined in the intent filter before an activity can be invoked.

Using Intent filter

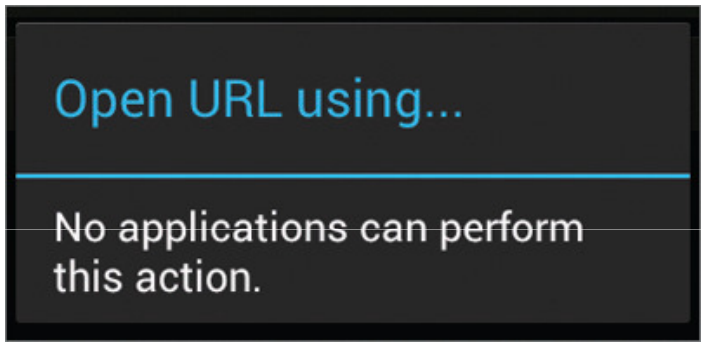
- Declare in AndroidManifest.xml

```
<activity android:name=".MyBrowserActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="net.learn2develop.MyBrowser" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="net.learn2develop.Apps" />
        <category android:name="net.learn2develop.OtherApps" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

Using Intent Filters

- Add multiple categories to an Intent object

```
public void onClickLaunchMyBrowser(View view) {
    /*
    Intent i = new
            Intent("net.learn2develop.MyBrowser");
    i.setData(Uri.parse("http://www.amazon.com"));
    startActivity(i);
    */

    Intent i = new
            Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("http://www.amazon.com"));
    //i.addCategory("net.learn2develop.Apps");
    //---this category does not match any in the intent-filter---
    i.addCategory("net.learn2develop.OtherApps");
    i.addCategory("net.learn2develop.SomeOtherApps");
    startActivity(Intent.createChooser(i, "Open URL using..."));
}
```



Try it out

- Page 54
- Page 59
- Page 63
- Page 85
- Page 91

Notification

- A notification is a message you can display to the user outside of your application's normal UI.
- When you tell the system to issue a notification, it first appears as an icon in the **notification area**.
- To see the details of the notification, the user opens the **notification drawer**.

Notification

Figure 1. Notifications in the notification area.

Figure 2. Notifications in the notification drawer.

Normal View

- The callouts in the illustration refer to the following:

1. Content title
2. Large icon
3. Content text
4. Content info
5. Small icon
6. Time that the notification was issued. You can set an explicit value with setWhen(); if you don't, it defaults to the time that the system received the notification.

- http://developer.android.com/guide/topics/ui/notifiers/notifications.html

Building a Notification

1. Create a Notification Builder
2. Define the Notification's Action
3. Set the Notification's Click Behavior
4. Issue the Notification

- NotificationCompat.Builder (in the support library)

Creating a notification

- Specify the UI information and actions for a notification in a NotificationCompat.Builder object.
- Create the notification itself, call NotificationCompat.Builder.build(), which returns a Notification object containing your specifications.
- Issue the notification, pass the Notification object to the system by calling NotificationManager.notify()

Creating a Notification

- A Notification object *must* contain the following:
  - A small icon, set by setSmallIcon()
  - A title, set by setContentTitle()
  - Content text, set by setContentText()
- All other notification settings and contents are optional.
- To learn more about them, see the reference documentation for NotificationCompat.Builder.

Creating a notification

- You should always define the action that's triggered when the user clicks the notification; usually this action opens an Activity in your application.
- Other issue: http://developer.android.com/guide/topics/ui/notifiers/notifications.html#CreateNotification
- The following snippet illustrates a simple notification that specifies an activity to open when the user clicks the notification.

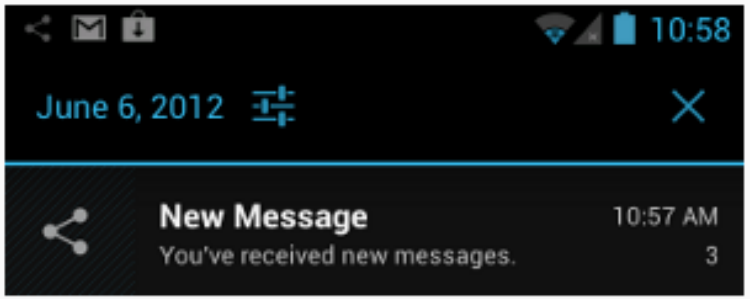
Creating a notification

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());
```

Updating notifications

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
...
```

This produces a notification that looks like this:



Removing notifications

- Notifications remain visible until one of the following happens:
  - The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
  - The user clicks the notification, and you called setAutoCancel() when you created the notification.
  - You call cancel() for a specific notification ID. This method also deletes ongoing notifications.
  - You call cancelAll(), which removes all of the notifications you previously issued.

Example: Displaying notifications on the status bar

- Button to be clicked to display Notification

```
<Button
    android:id="@+id/btn_displaynotif"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Display Notification"
    android:onClick="onClick"/>
```

Codes to display notification

```
protected void displayNotification()
{
    //---PendingIntent to launch activity if the user selects
    // this notification---
    Intent i = new Intent(this, NotificationView.class);
    i.putExtra("notificationID", notificationID);

    PendingIntent pendingIntent =
        PendingIntent.getActivity(this, 0, i, 0);

    NotificationManager nm = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);

    Notification notif = new Notification(
        R.drawable.ic_launcher,
        "Reminder: Meeting starts in 5 minutes",
        System.currentTimeMillis());

    CharSequence from = "System Alarm";
    CharSequence message = "Meeting with customer at 3pm...";

    notif.setLatestEventInfo(this, from, message, pendingIntent);

    //---100ms delay, vibrate for 250ms, pause for 100 ms and
    // then vibrate for 500ms---
    notif.vibrate = new long[] { 100, 250, 100, 500};
    nm.notify(notificationID, notif);
}
```

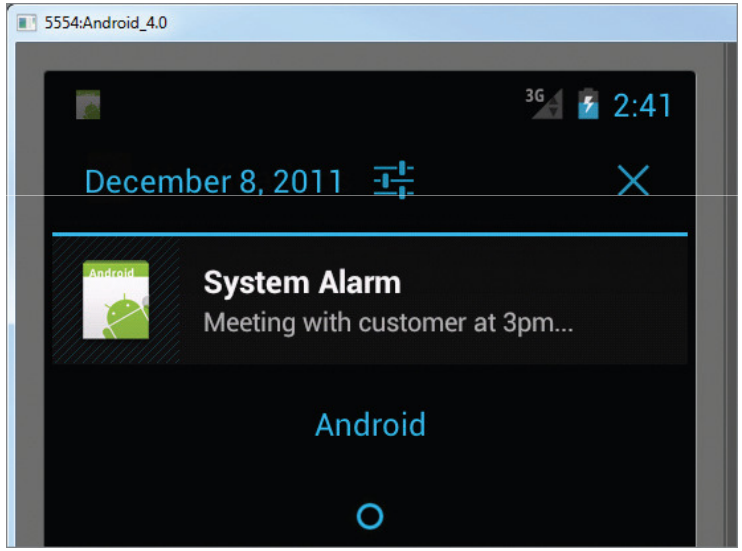
created an Intent object to point to the NotificationView class

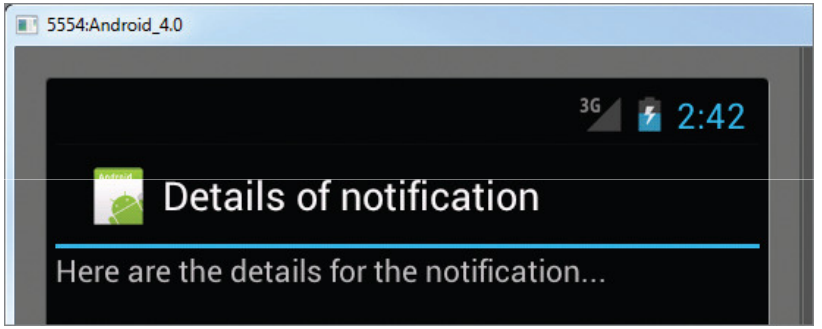
A PendingIntent object helps you to perform an action on your application's behalf, often at a later time, regardless of whether your application is running or not.

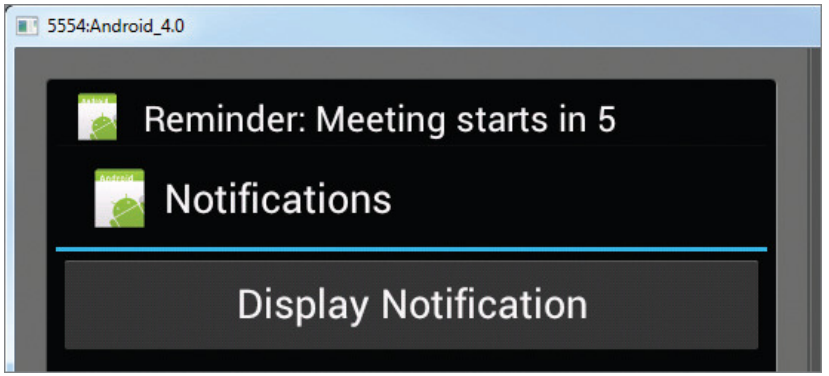
The Notification class enables you to specify the notification's main information when the notification first appears on the status bar. The second argument to the Notification constructor sets the "ticker text" on the status bar

set the details of the notification using the setLatestEventInfo()

Notifications







Notification

- On the Notification View

```
//---look up the notification manager service---
NotificationManager nm = (NotificationManager)
    getSystemService(NOTIFICATION_SERVICE);

//---cancel the notification that we started---
nm.cancel(getIntent().getExtras().getInt("notificationID"));
```

When the user clicks on the notification, the NotificationView activity is launched. Here, you dismiss the notification by using the cancel() method of the NotificationManager object and passing it the ID of the notification (passed in via the Intent object)

Try it out

- Page 98