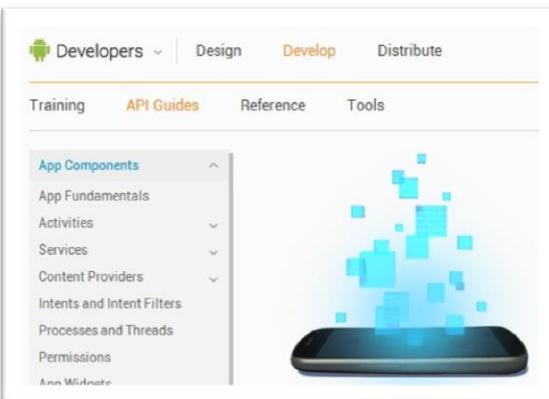


# Course Overview



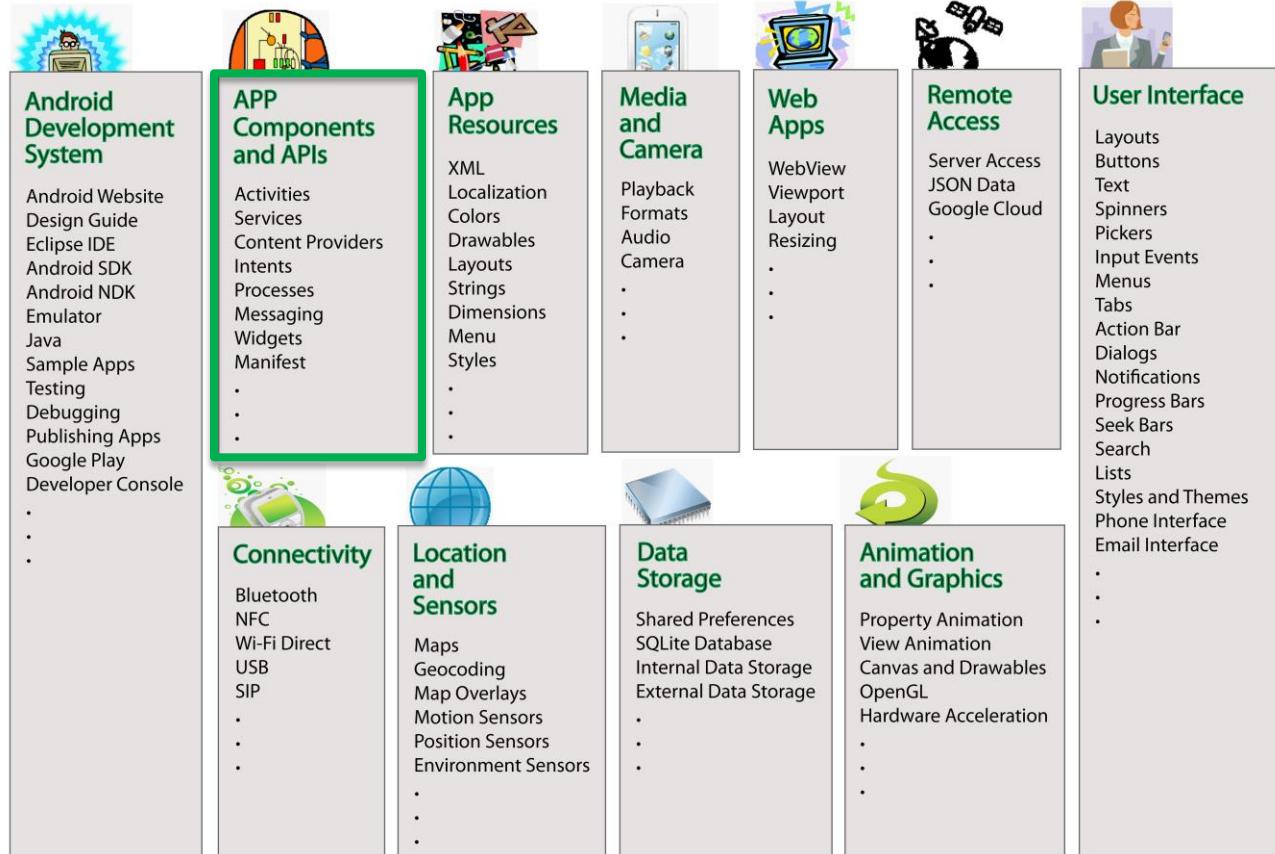
Android Developers Website



The screenshot shows the main navigation bar with links for Developers, Design, Develop, and Distribute. Below the navigation bar, there are tabs for Training, API Guides (which is selected), Reference, and Tools. On the left, a sidebar menu is open under 'App Components', listing items like App Fundamentals, Activities, Services, Content Providers, Intent Filters, Processes and Threads, and Permissions. A large image of a smartphone with floating blue squares is displayed in the center.

<http://developer.android.com/guide/components/index.html>

Google Play is a trademark of Google Inc.

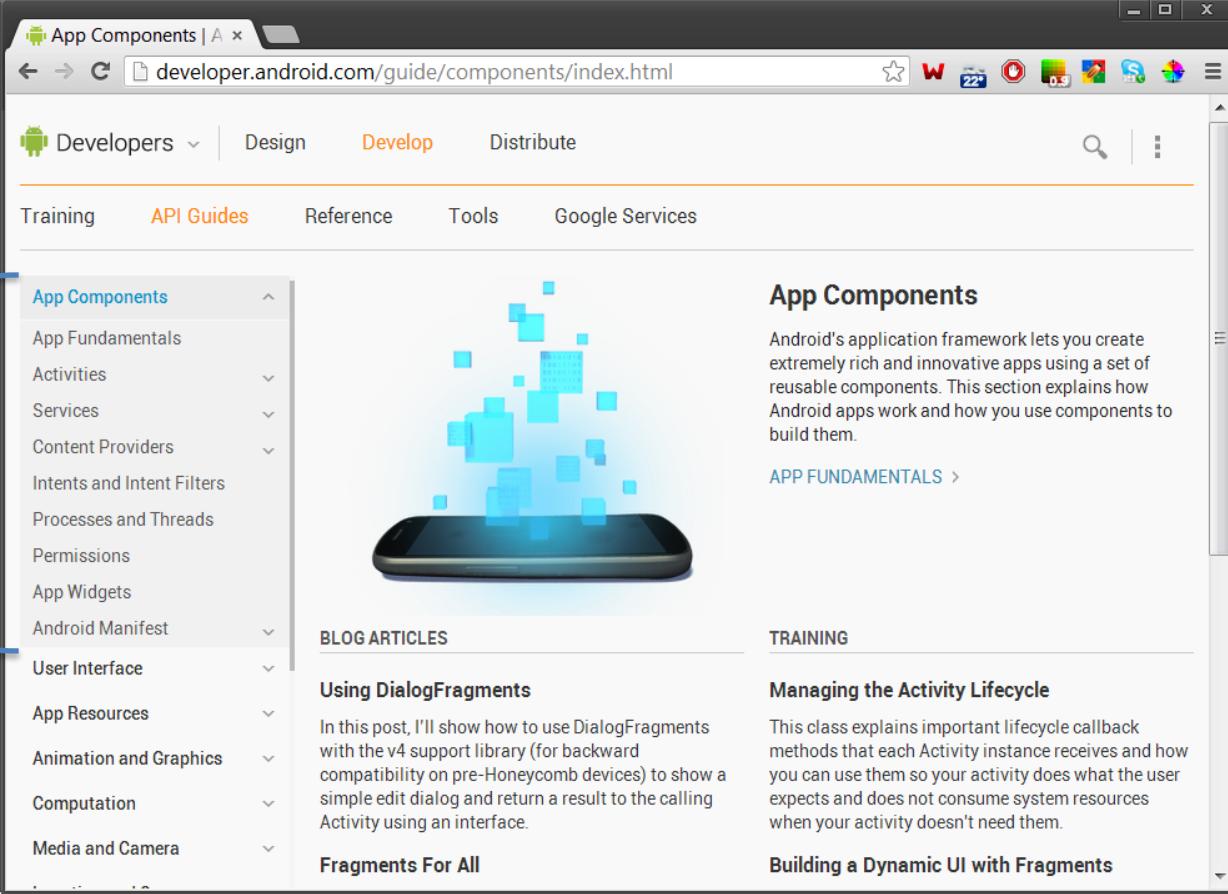


# App Components and APIs

The basic building blocks of app actions.

<http://developer.android.com/guide/components/index.html>

Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).



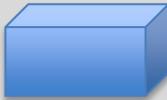
A screenshot of a web browser displaying the 'App Components' section of the Android Developers website at developer.android.com/guide/components/index.html. The browser window has a dark theme. At the top, there's a navigation bar with tabs for 'Design', 'Develop', and 'Distribute'. Below the tabs, there are links for 'Training', 'API Guides' (which is highlighted in orange), 'Reference', 'Tools', and 'Google Services'. On the left, a sidebar titled 'App Components' lists various components: App Fundamentals, Activities, Services, Content Providers, Intents and Intent Filters, Processes and Threads, Permissions, App Widgets, Android Manifest, User Interface, App Resources, Animation and Graphics, Computation, and Media and Camera. A blue curly brace on the left side groups the 'The basic building blocks of app actions.' text with the 'App Components' sidebar. The main content area features a large image of a smartphone with a blue glowing effect around it, and the heading 'App Components'. Below the heading, a paragraph explains that Android's application framework lets you create extremely rich and innovative apps using a set of reusable components. It also includes a 'APP FUNDAMENTALS >' link. Further down, there are sections for 'BLOG ARTICLES' (with 'Using DialogFragments') and 'TRAINING' (with 'Managing the Activity Lifecycle').

# App Components and APIs in Use

Android Application

Resources

- XML
- Images



Components

- Activities
- Content Providers
- Services
- Broadcast Receivers



APIs

- Classes
- Methods

- Fragments
- Widgets
- Processes & Threads
- Messaging
- Memory Management
- Permissions
- ...



APK Upload

Google Play

APK Download



App Development

APK Creation



Device Supplier Load



By Smieh (This file was derived from: Sample.svg) [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>)], via Wikimedia Commons

# Android Architecture

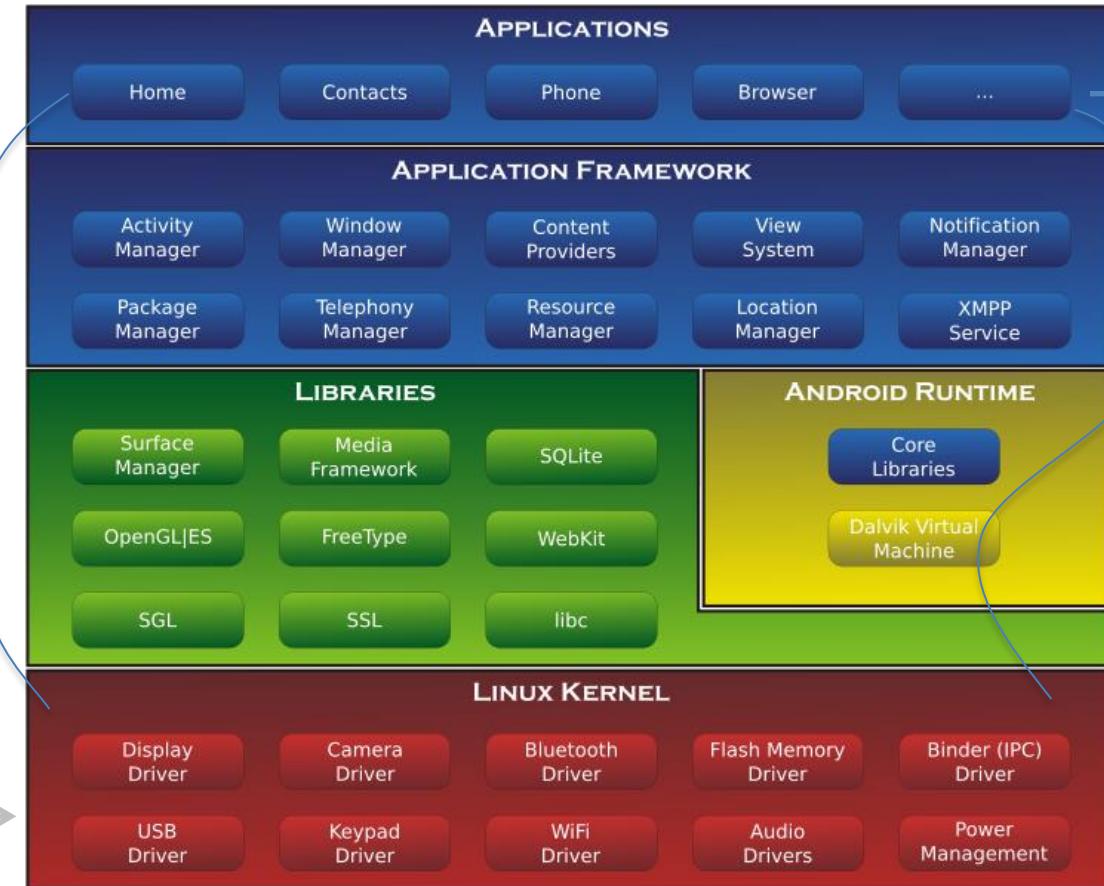
Multi-user Linux System

Permissions are set for all files in an app so that only the user ID assigned to that app can access them.

Each app is a separate Linux user with its own Linux user ID (unknown to the app).

<http://developer.android.com/guide/components/fundamentals.html>

[http://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)#Development](http://en.wikipedia.org/wiki/Android_(operating_system)#Development)



Each app process has its own Dalvik Virtual Machine, which runs in its own Linux process.

By Smieh (This file was derived from: Sample.svg) [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>)], via Wikimedia Commons

# Android Linux

Linux is open source and was initially developed in the 1990's:  
<http://en.wikipedia.org/wiki/Linux>

Linux is a branch of the Unix operating system:  
[http://en.wikipedia.org/wiki/File:Unix\\_history.svg](http://en.wikipedia.org/wiki/File:Unix_history.svg)

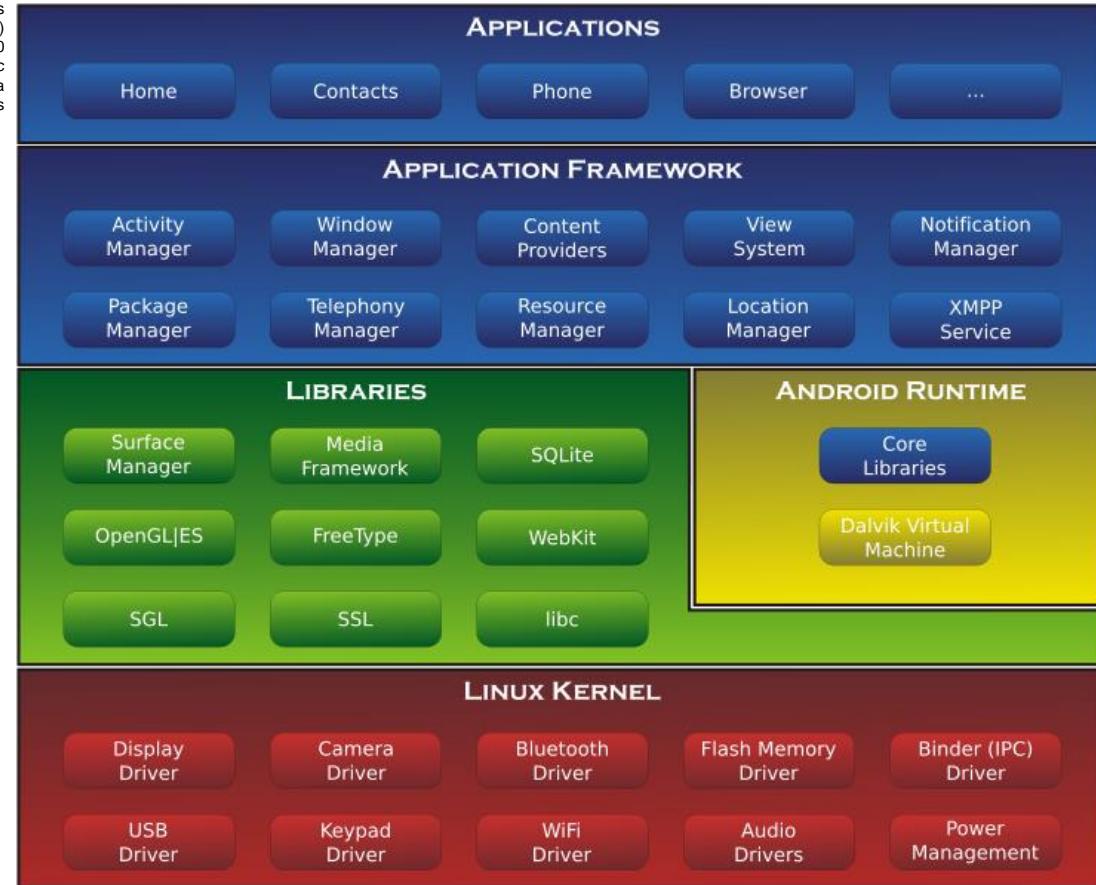
Some Android changes to the Linux Kernel:  
[http://elinux.org/Android\\_Kernel\\_Features](http://elinux.org/Android_Kernel_Features)

- Scheduling
- Time Sharing
- Prioritization
- Code Execution

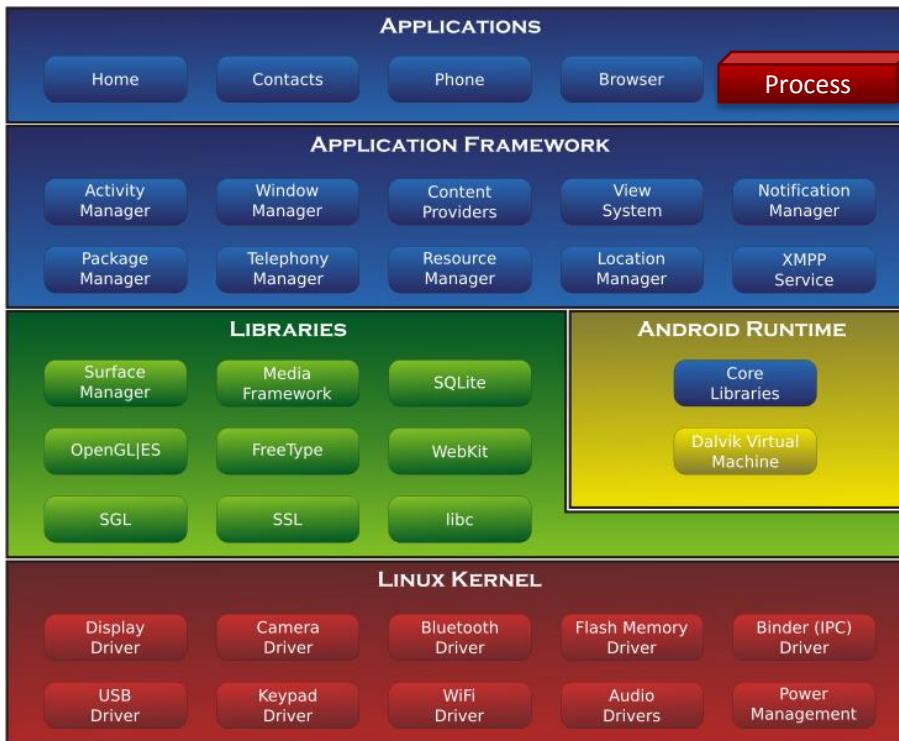
## Process

<http://oreilly.com/catalog/linuxkernel/chapter/ch10.html>

By Smieh (This file was derived from: Sample.svg)  
[CC-BY-SA-3.0  
(<http://creativecommons.org/licenses/by-sa/3.0/>), via Wikimedia Commons



# App Lifecycles



## L i f e c y c l e P h a s e s

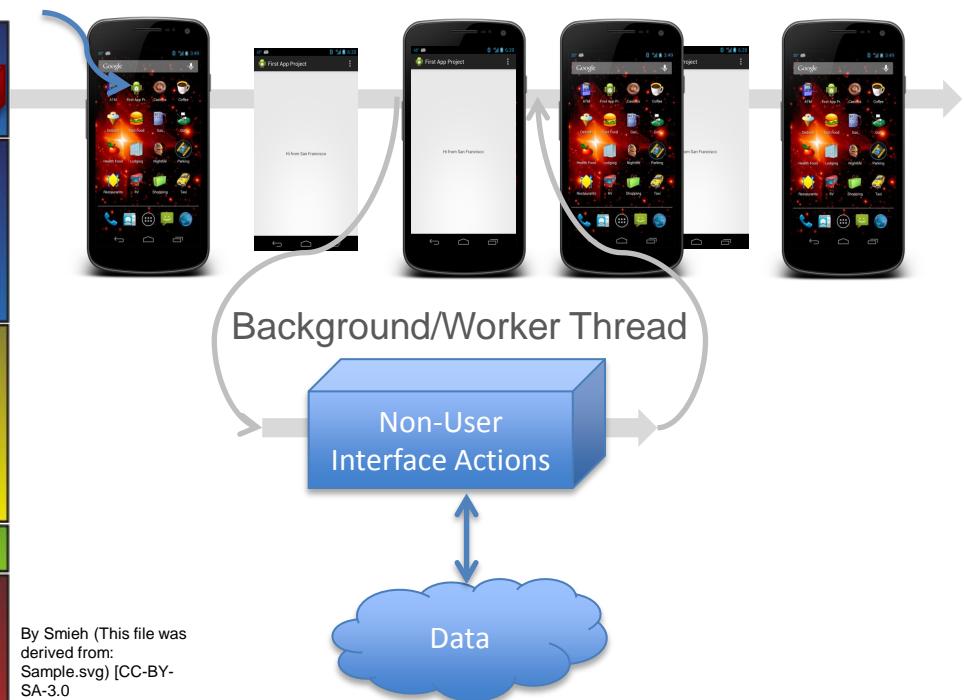
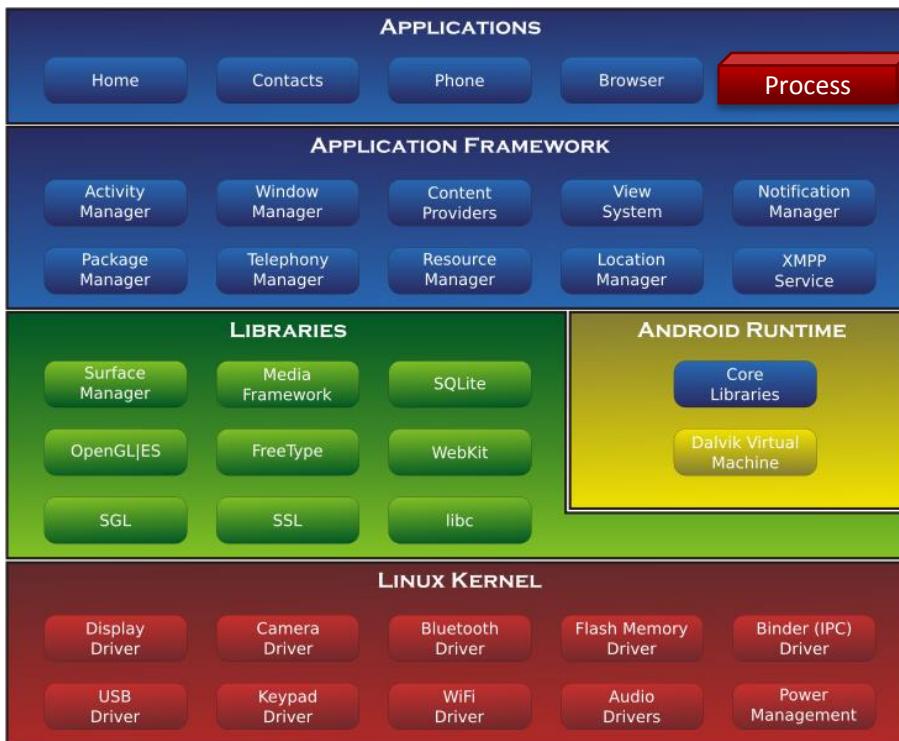
### Lifecycle Patterns

- Activities
- Services
- Content Providers
- Broadcast Receivers

### Lifecycle Methods

```
onPhase1() { . . . }
onPhase2() { . . . }
onPhase3() { . . . }
```

# App Threads



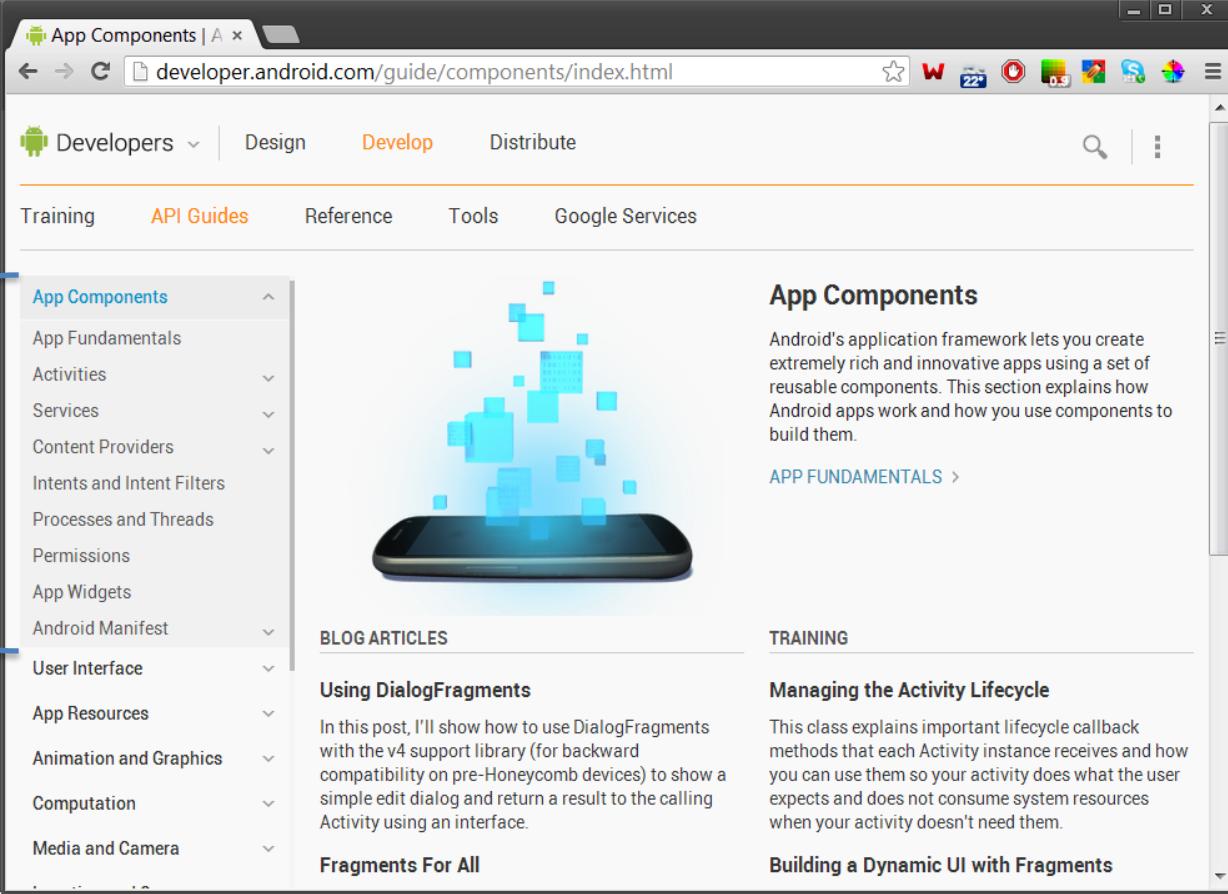
By Smieh (This file was derived from: Sample.svg) [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

# App Components and APIs

The basic building blocks of app actions.

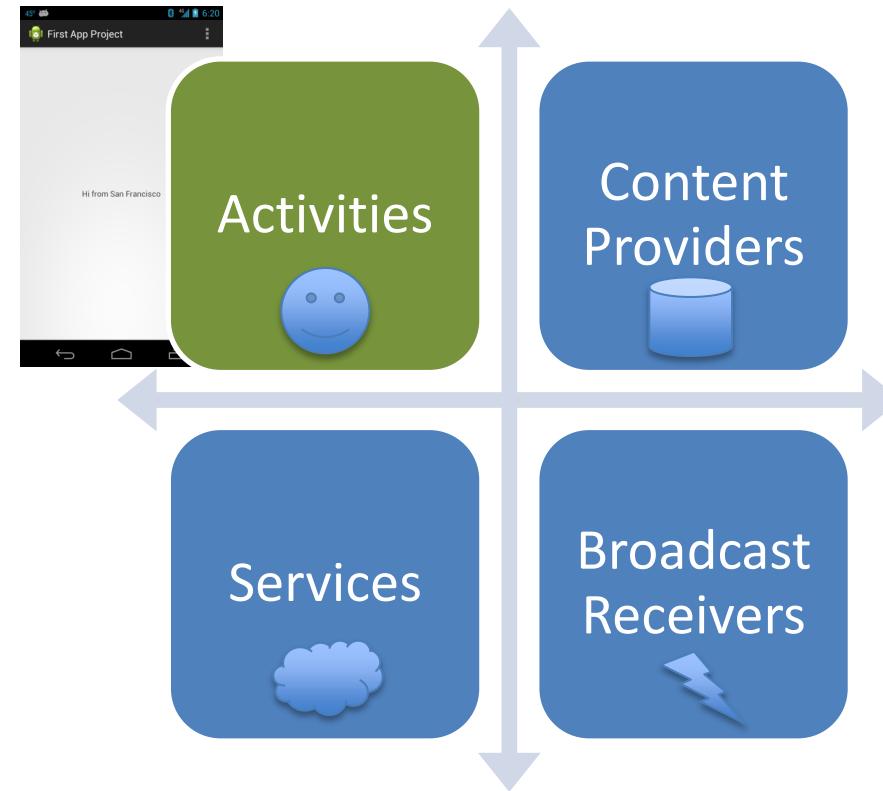
<http://developer.android.com/guide/components/index.html>

Portions of this page are reproduced from work created and shared by the [Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).



A screenshot of a web browser displaying the 'App Components' section of the Android Developers website at developer.android.com/guide/components/index.html. The browser window has a title bar 'App Components | A x' and a URL bar showing the same address. The main content area features a navigation bar with tabs: 'Developers' (selected), 'Design', 'Develop' (highlighted in orange), and 'Distribute'. Below the tabs are links for 'Training', 'API Guides' (highlighted in orange), 'Reference', 'Tools', and 'Google Services'. On the left, a sidebar titled 'App Components' lists various components: App Fundamentals, Activities, Services, Content Providers, Intents and Intent Filters, Processes and Threads, Permissions, App Widgets, Android Manifest, User Interface, App Resources, Animation and Graphics, Computation, and Media and Camera. A blue curly brace on the left side groups the 'The basic building blocks of app actions.' text with the 'App Components' sidebar. The main content area contains a large image of a smartphone with a blue glowing effect around it, and text explaining 'App Components' and linking to 'APP FUNDAMENTALS'. Below this are sections for 'BLOG ARTICLES' (Using DialogFragments, Fragments For All) and 'TRAINING' (Managing the Activity Lifecycle, Building a Dynamic UI with Fragments).

# Activities



# Activities API

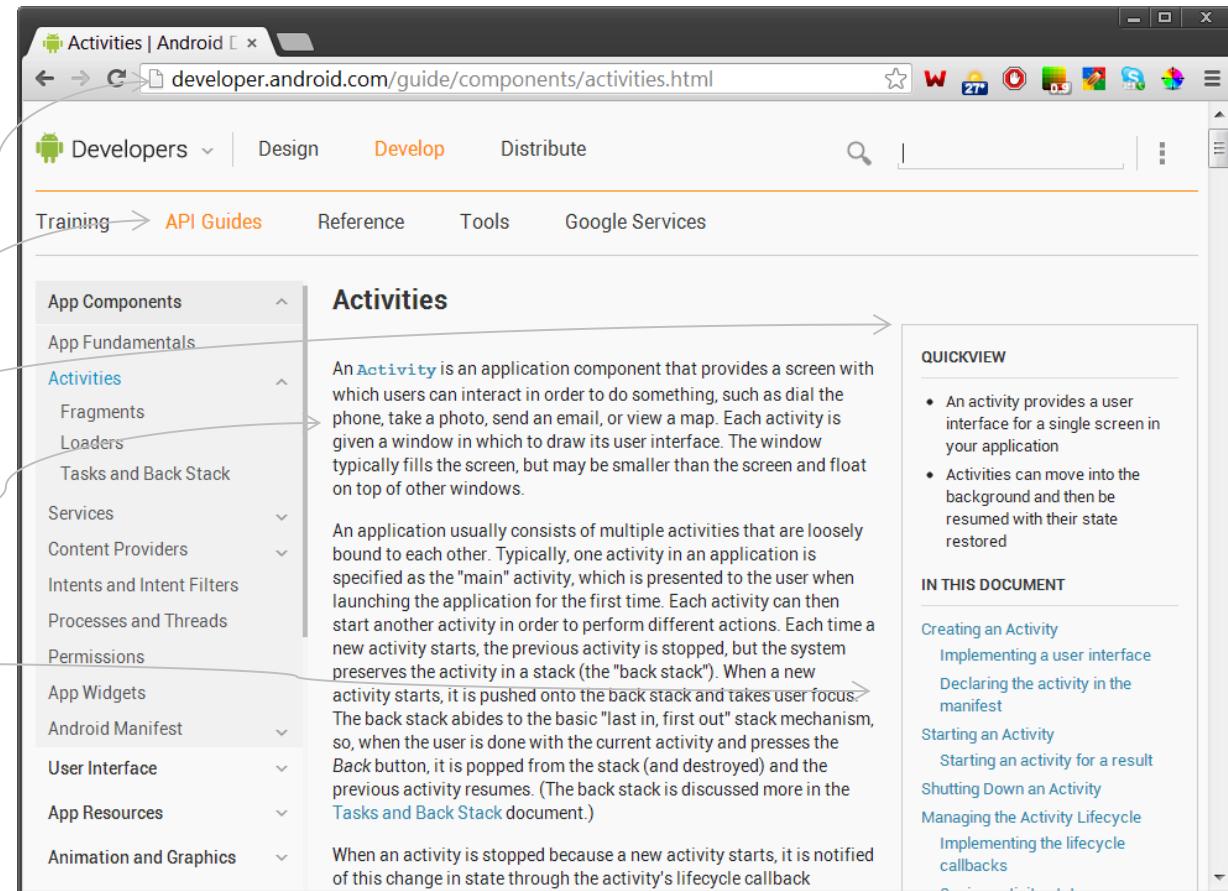
<http://developer.android.com/guide/components/activities.html>

Development API Guide

Overview

Basic Description

Detailed Description



The screenshot shows a web browser displaying the Android Developers website at developer.android.com/guide/components/activities.html. The page is titled 'Activities' under the 'App Components' section. The left sidebar lists various app components: App Fundamentals, Activities (which is currently selected), Fragments, Loaders, Tasks and Back Stack, Services, Content Providers, Intents and Intent Filters, Processes and Threads, Permissions, App Widgets, Android Manifest, User Interface, App Resources, and Animation and Graphics. The main content area describes what an Activity is, how multiple activities are typically bound together, and the back stack mechanism. A 'QUICKVIEW' sidebar on the right contains bullet points about activities providing user interfaces and moving between states. Another sidebar on the right lists related topics like 'Creating an Activity' and 'Starting an Activity'.

An **Activity** is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the *Back* button, it is popped from the stack (and destroyed) and the previous activity resumes. (The back stack is discussed more in the [Tasks and Back Stack](#) document.)

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback

**QUICKVIEW**

- An activity provides a user interface for a single screen in your application
- Activities can move into the background and then be resumed with their state restored

**IN THIS DOCUMENT**

[Creating an Activity](#)  
[Implementing a user interface](#)  
[Declaring the activity in the manifest](#)  
[Starting an Activity](#)  
[Starting an activity for a result](#)  
[Shutting Down an Activity](#)  
[Managing the Activity Lifecycle](#)  
[Implementing the lifecycle callbacks](#)

# First App Activity

Extends Activity Class

```
package com.donkcowan.firstappproject;  
import android.os.Bundle;  
import android.app.Activity;  
import android.util.Log;  
import android.view.Menu;
```

public class MainActivity extends Activity {

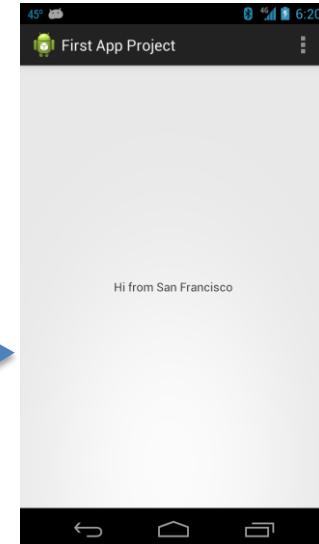
```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    int variableTest = 1;  
    int variableUse = variableTest;  
    Log.i("AppTest","My message");  
    Log.i("AppTest",Integer.toString(variableUse));  
}
```

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.activity_main, menu);  
    return true;  
}
```



L i f e c y c l e   P h a s e s

A c t i v i t y   M e t h o d s



# Activity Class

[http://developer.android.com  
/reference/android/app/Activity.html](http://developer.android.com/reference/android/app/Activity.html)

Development Reference

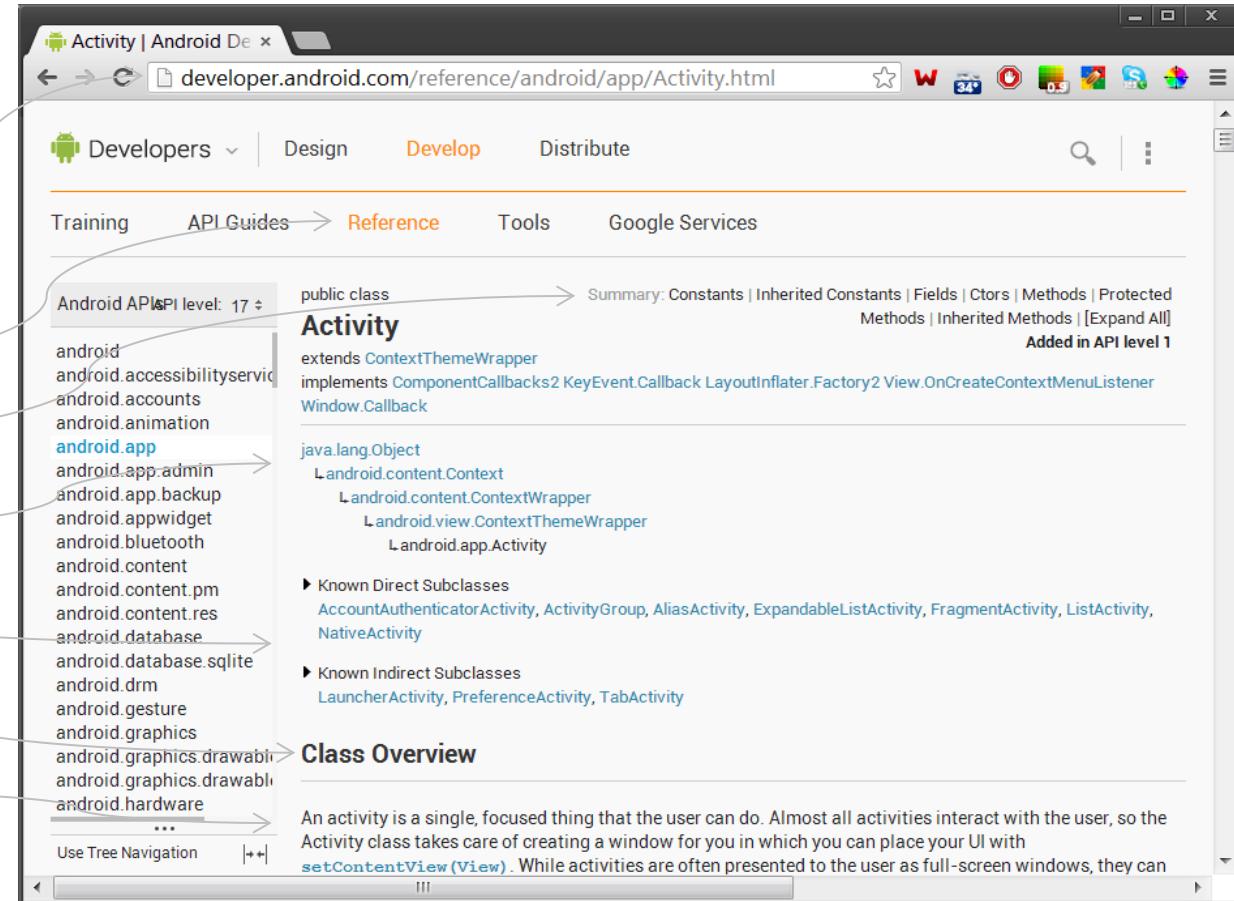
Summary Links

Hierarchy of Class Extensions

Subclasses

Class Overview

*"An activity is a single, focused thing that the user can do."*



The screenshot shows a web browser displaying the `Activity` class documentation from the [Android Developers site](http://developer.android.com). The URL in the address bar is `developer.android.com/reference/android/app/Activity.html`. The page is part of the **Reference** section, indicated by the orange navigation bar.

The main content area shows the `Activity` class definition:

```
public class Activity
    extends ContextThemeWrapper
    implements ComponentCallbacks2, KeyEvent.Callback, LayoutInflator.Factory2, View.OnCreateContextMenuListener, Window.Callback
```

It also lists the class's inheritance chain:

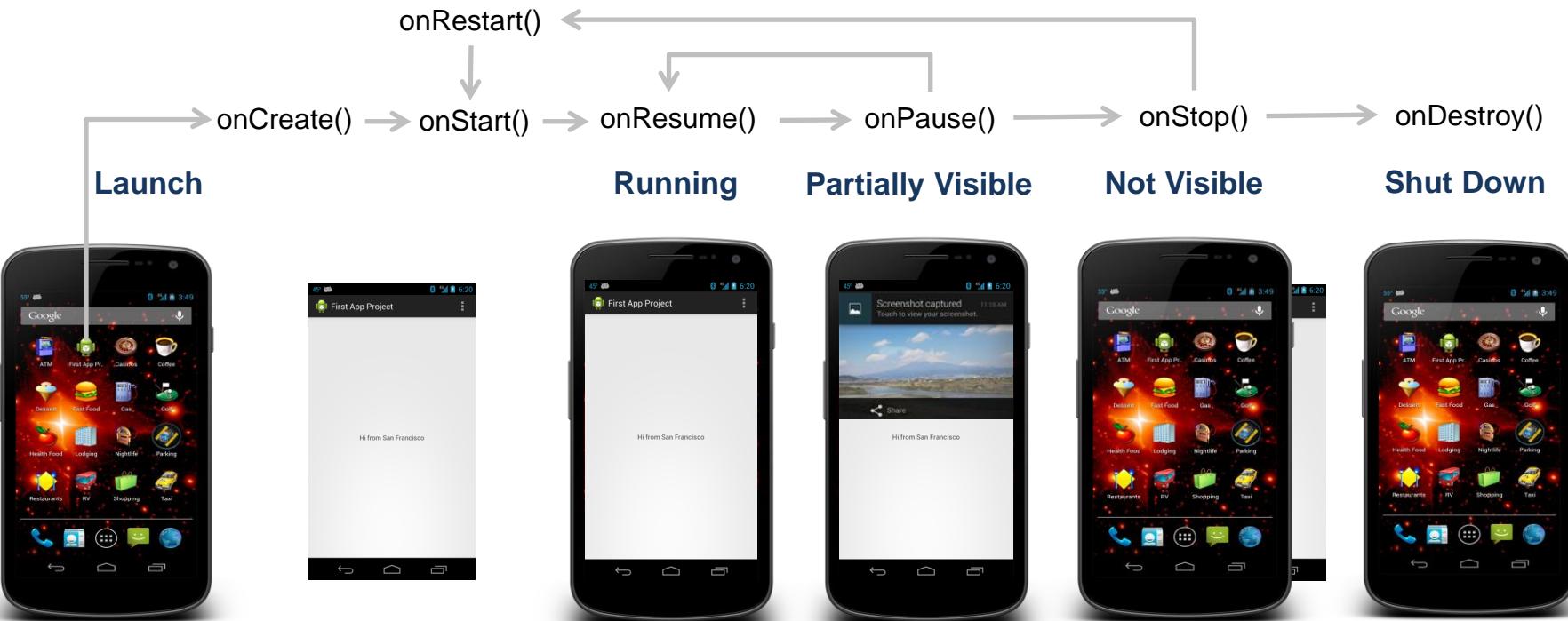
```
java.lang.Object
└ android.content.Context
    └ android.content.ContextWrapper
        └ android.view.ContextThemeWrapper
            └ android.app.Activity
```

Below this, there are sections for **Known Direct Subclasses** (e.g., `AccountAuthenticatorActivity`, `ActivityGroup`, `AliasActivity`, `ExpandableListActivity`, `FragmentActivity`, `ListActivity`, `NativeActivity`) and **Known Indirect Subclasses** (e.g., `LauncherActivity`, `PreferenceActivity`, `TabActivity`).

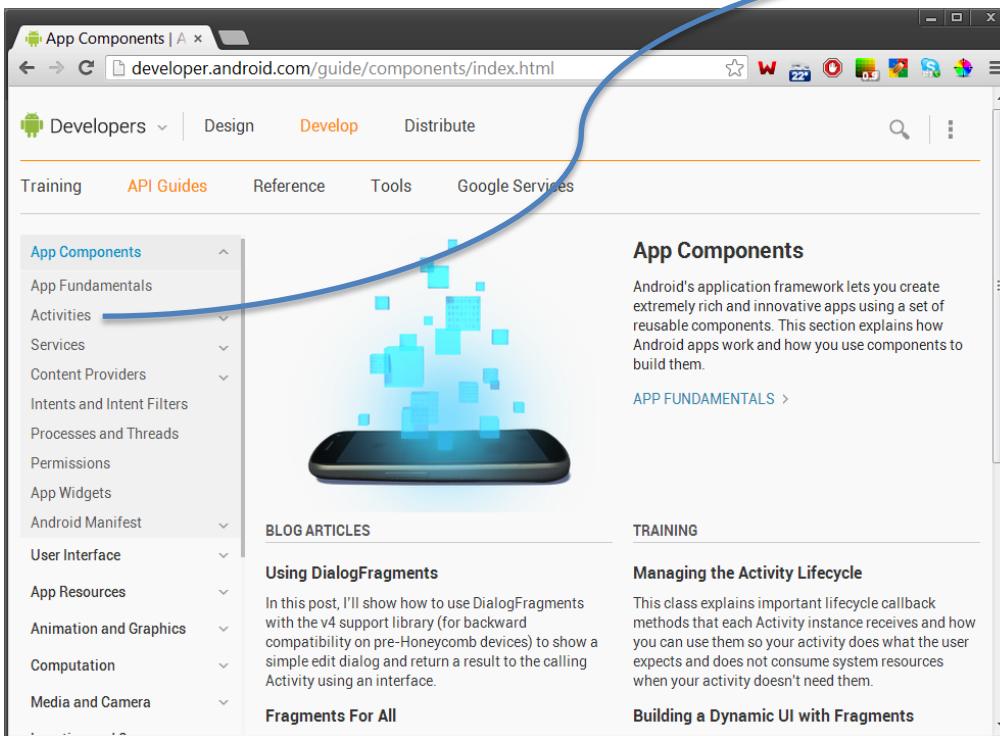
The **Class Overview** section contains the following text:

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the `Activity` class takes care of creating a window for you in which you can place your UI with `setContentView(View)`. While activities are often presented to the user as full-screen windows, they can

# Activity Lifecycle



# Activity Lifecycle



The screenshot shows a web browser displaying the Android developer documentation for App Components. The sidebar on the left has 'Activities' selected under 'App Components'. The main content area is titled 'App Components' and discusses how to create rich apps using reusable components like Activities.

**App Components**

Android's application framework lets you create extremely rich and innovative apps using a set of reusable components. This section explains how Android apps work and how you use components to build them.

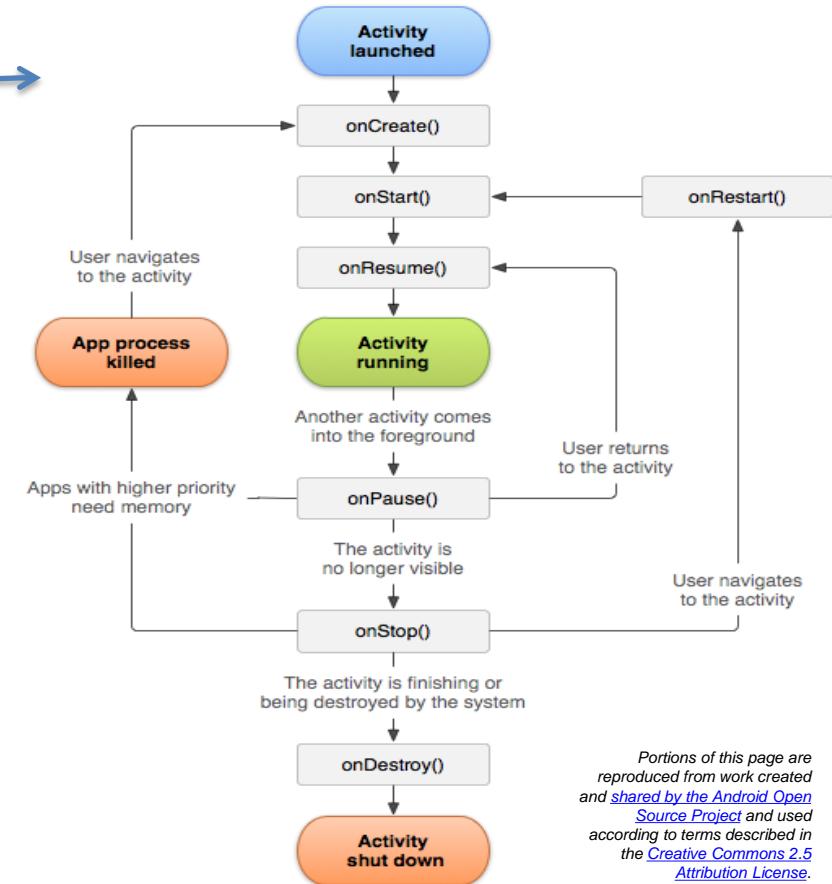
[APP FUNDAMENTALS >](#)

**BLOG ARTICLES**

- [Using DialogFragments](#)
- [Managing the Activity Lifecycle](#)

**TRAINING**

- [Animation and Graphics](#)
- [Computation](#)
- [Media and Camera](#)
- [Fragments For All](#)



Portions of this page are reproduced from work created and shared by the [Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).

# Activity Lifecycle Methods and Typical Uses

## **onCreate()**

- Must be implemented
- Call setContentView()
- Initialize display layout
- Initialize variables
- Start background thread

## **onResume()**

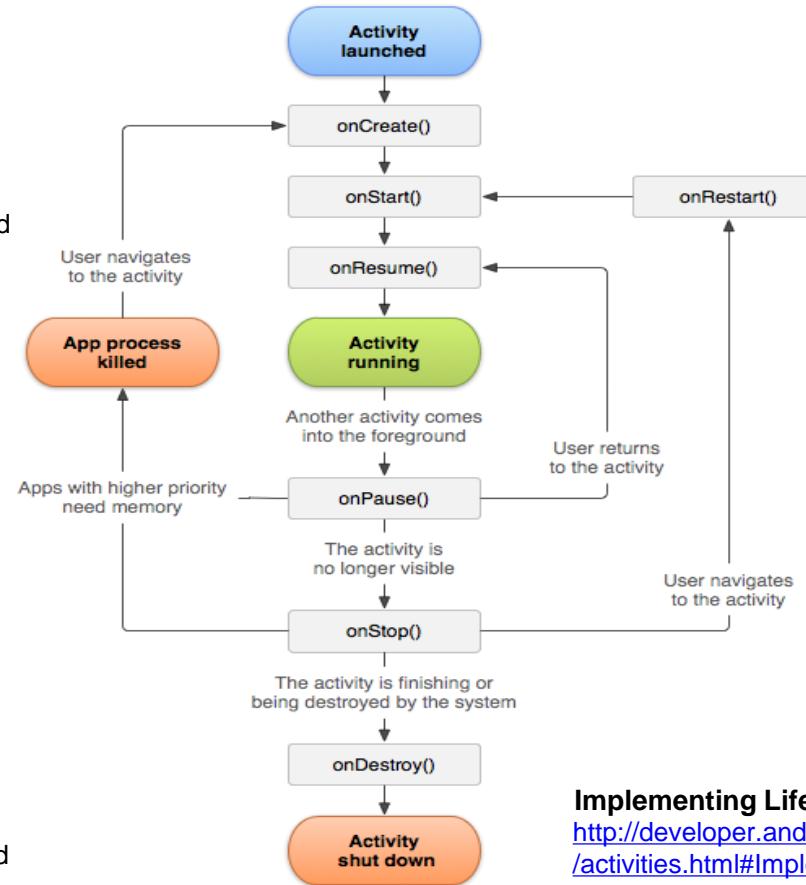
- Restore variables
- Restart animations
- Lightweight code

## **onPause()**

- Commit changes
- Save variables
- Save preferences
- Stop animations
- Lightweight code

## **onDestroy()**

- Close databases
- Release remaining resources
- Stop background thread



Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).

## **onStart()**

- Register Broadcast Receiver
- Reinitialize states

## **onRestart()**

- Open server connections
- Allocate resources

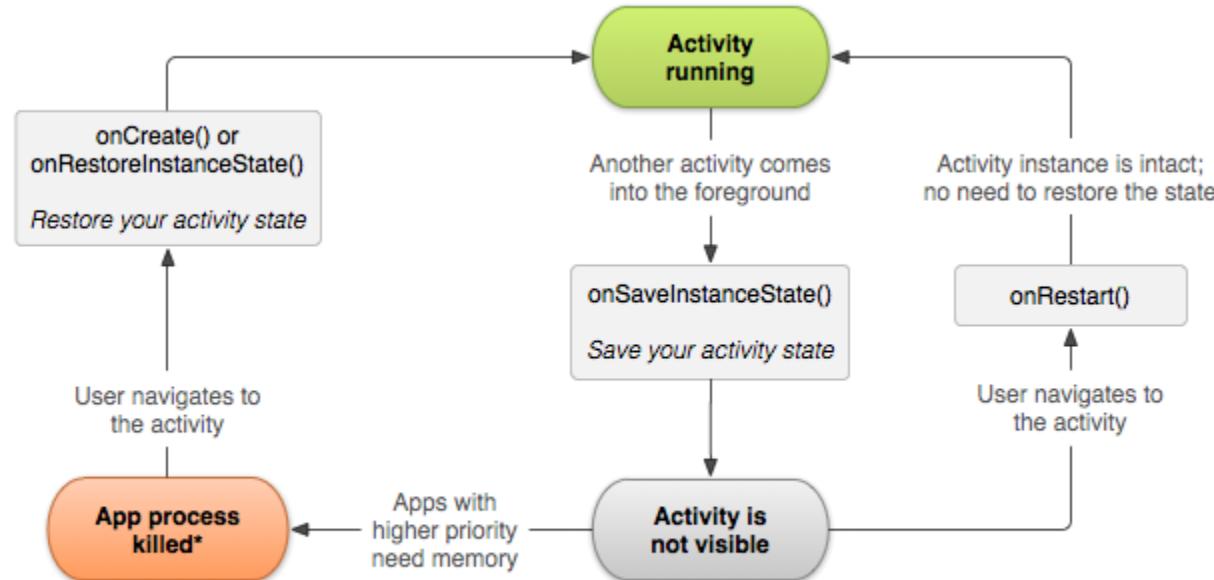
## **onStop()**

- Close server connections
- Free resources
- Unregister Broadcast Receiver

## Implementing Lifecycle Callbacks

<http://developer.android.com/guide/components/activities.html#ImplementingLifecycleCallbacks>

# Saving Activity State



\*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

# Understanding an API

## Familiarity with major aspects

The screenshot shows a web browser displaying the Android developer guide for Activities. The left sidebar contains a navigation tree under 'App Components' with sections like App Fundamentals, Activities, Services, Content Providers, Intent and Intent Filters, Processes and Threads, Permissions, App Widgets, and more. The main content area is titled 'Activities' and defines an Activity as an application component providing a screen for user interaction. It also discusses the activity stack and lifecycle methods. On the right, there's a 'QUICKVIEW' section with links to creating and implementing activities, and a 'IN THIS DOCUMENT' section listing related topics. At the bottom right, there's a 'KEY CLASSES' section with a link to the Activity class.

Activities

An **Activity** is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the “main” activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the “back stack”). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic “last in, first out” stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes. (The back stack is discussed more in the Tasks and Back Stack document.)

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity’s lifecycle callback methods. There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides you the opportunity to perform specific work that’s appropriate to that state change. For instance, when stopped, your activity should release any large objects, such as network or database connections. When the activity resumes, you can reacquire the necessary resources and

**QUICKVIEW**

- An activity provides a user interface for a single screen of your application
- Activities can move into the background and then be resumed with their state restored

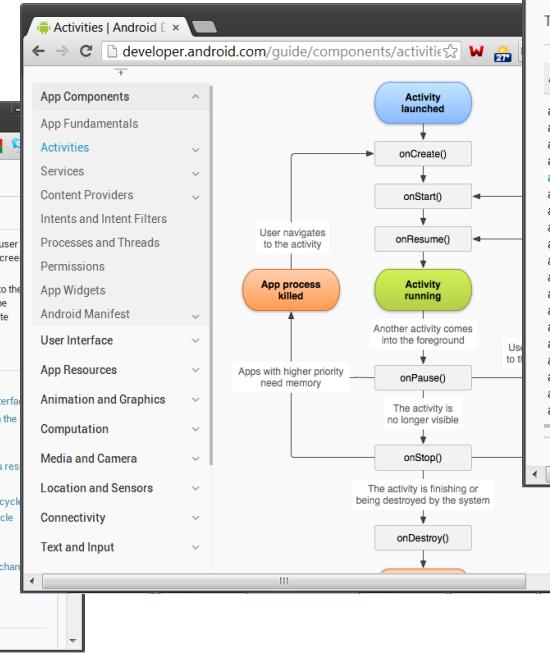
**IN THIS DOCUMENT**

[Creating an Activity](#)  
[Implementing a user interface](#)  
[Declaring the activity in the manifest](#)  
[Starting an Activity](#)  
[Starting an activity for a result](#)  
[Shutting Down an Activity](#)  
[Managing the Activity Lifecycle](#)  
[Implementing the lifecycle callbacks](#)  
[Saving activity state](#)  
[Handling configuration changes](#)  
[Coordinating activities](#)

**KEY CLASSES**

[Activity](#)

Understand key and required capabilities



Research as needed

Activity | Android De x developer.android.com/reference/android/z

Developers Design Develop Distribute

Training API Guides Reference Tools Google Services

Android API level: 17 ▾

public class **Activity**

Summary: Constants | Inherited Constants | Fields | Ctors | Methods | Protected Methods | Inherited Methods | [Expand All]  
Added in API level 1

extends ContextThemeWrapper

implements ComponentCallbacks2, KeyEvent.Callback, LayoutInflater.Factory2, View.OnCreateContextMenuListener, Window.Callback

View.OnCreateContextMenuItemListener

java.lang.Object

↳ android.content.Context

↳ android.content.ContextWrapper

↳ android.view.ContextThemeWrapper

↳ android.app.Activity

▶ Known Direct Subclasses

AccountAuthenticatorActivity, ActivityGroup, AliasActivity, ExpandableListActivity, FragmentActivity, ListActivity, NativeActivity

▶ Known Indirect Subclasses

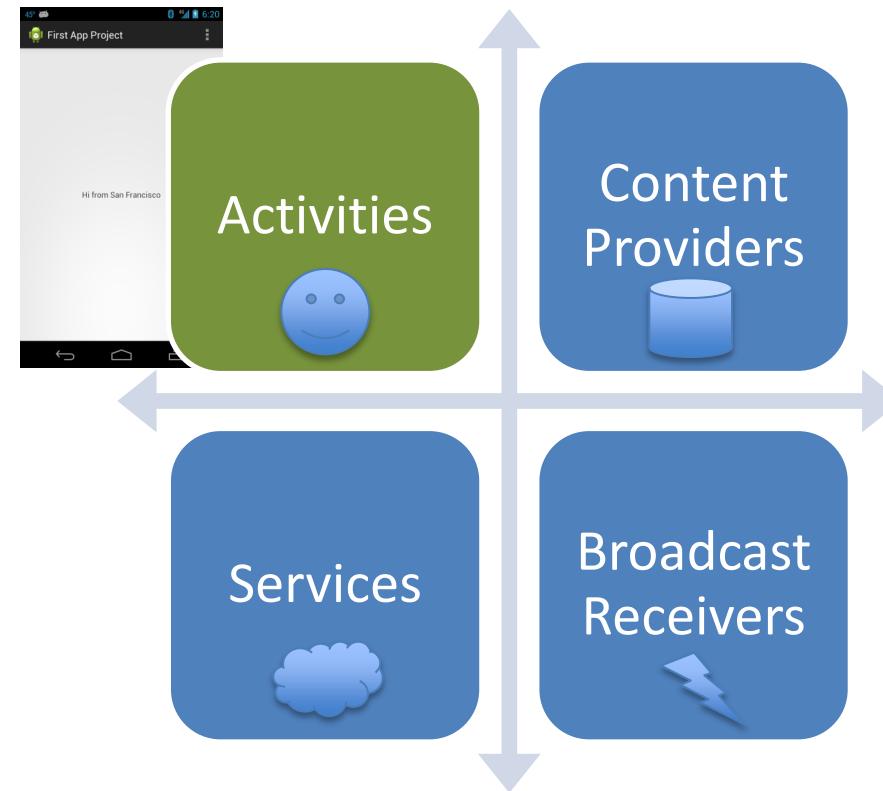
LauncherActivity, PreferenceActivity, TabActivity

**Class Overview**

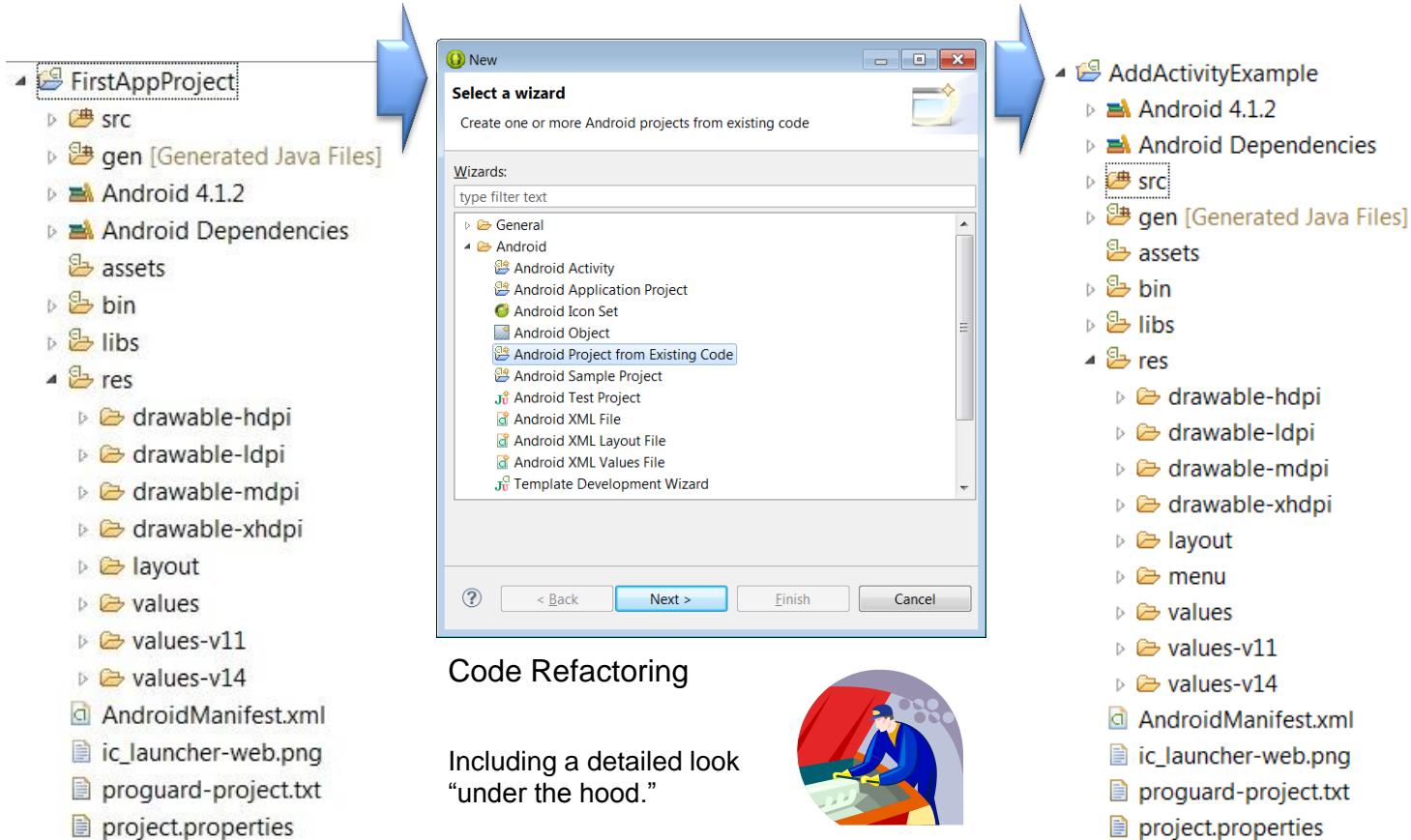
An activity is a single, focused thing that the user can do. Almost all activities

Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).

# Activities



# Creating a Project from Existing Code

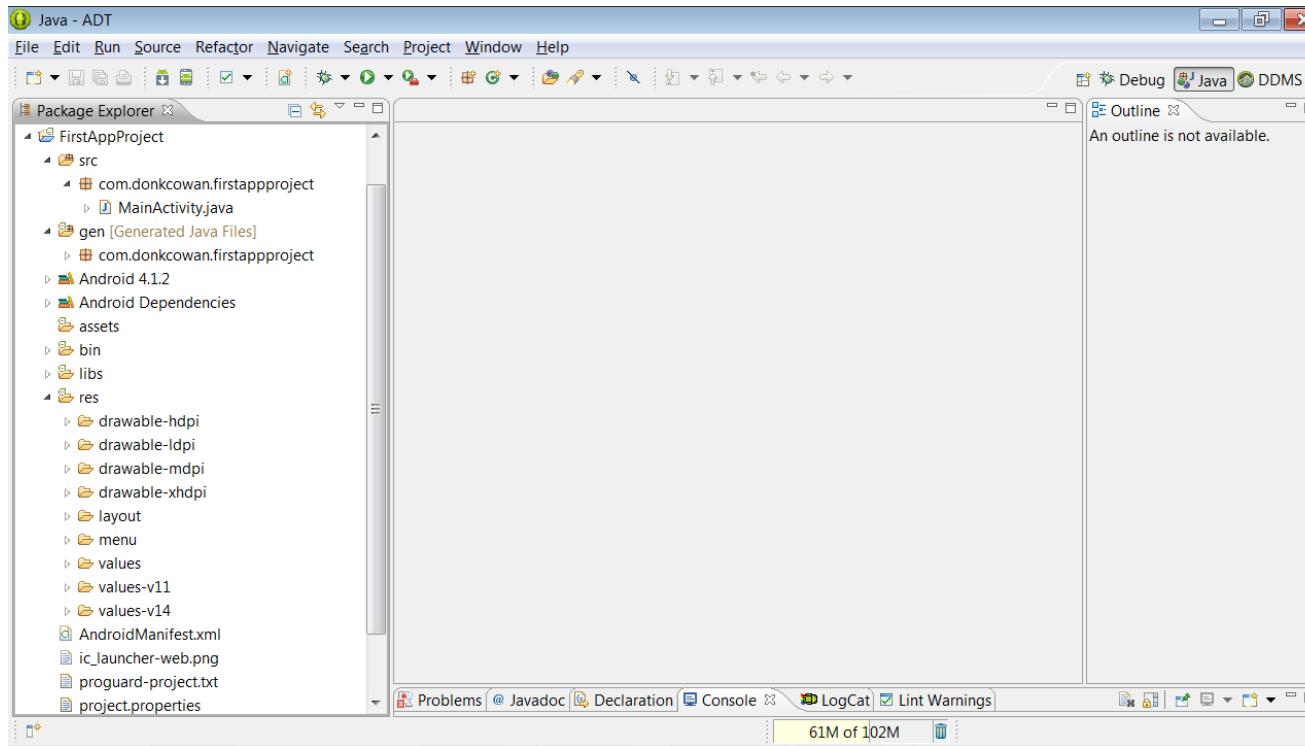


## Code Refactoring

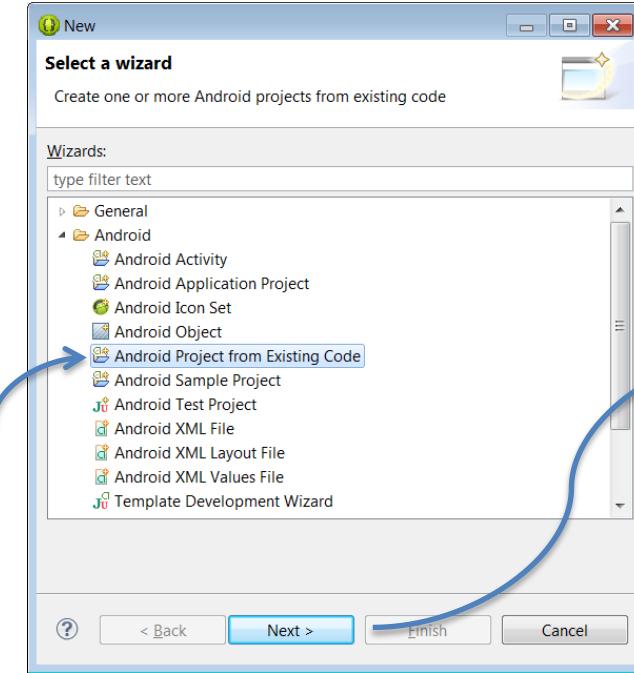
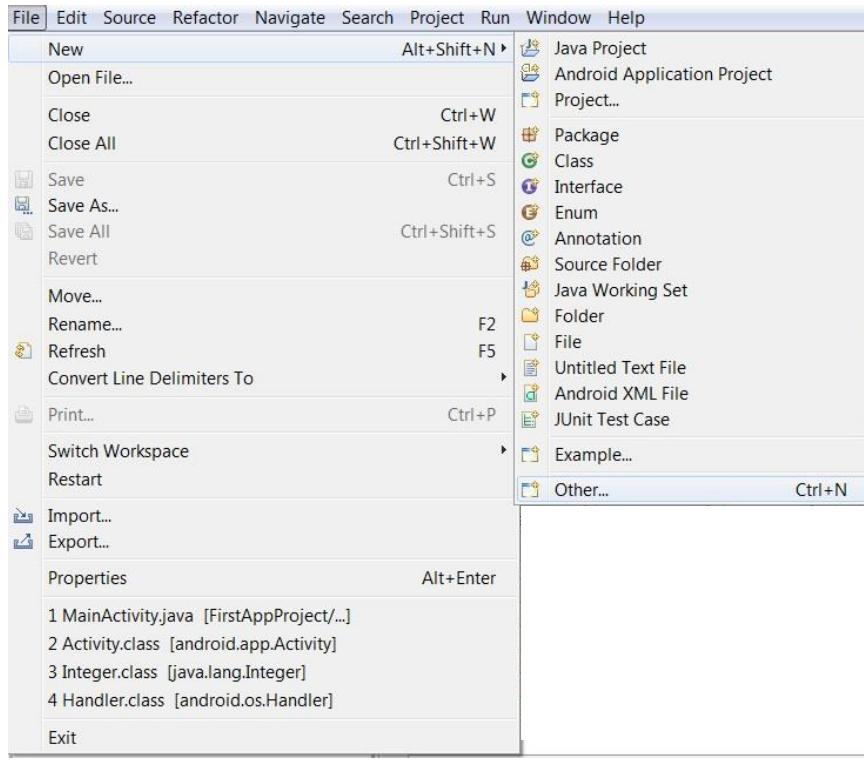
Including a detailed look  
“under the hood.”



# Opening Your Workspace



# Selecting the Project from Existing Code Wizard

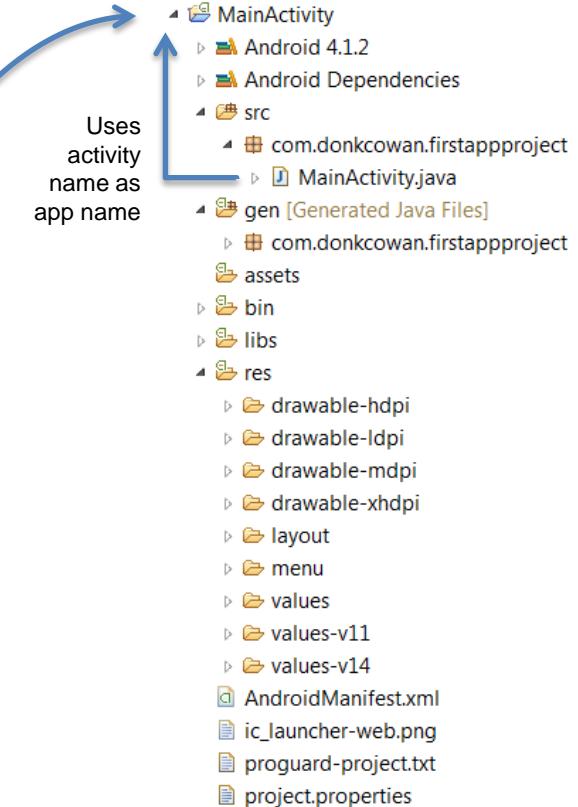
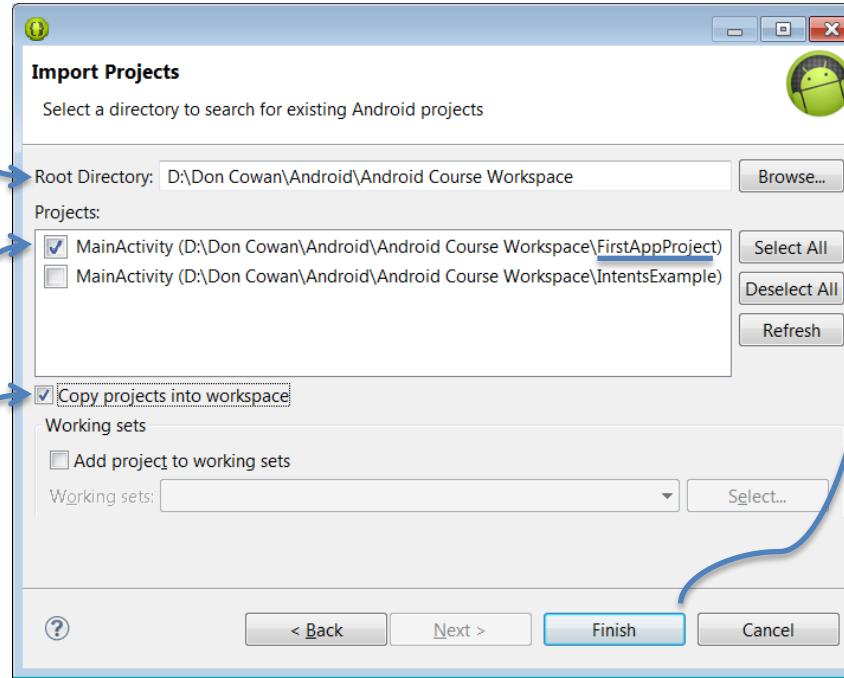


# Selecting the Project to Import

Select Workspace

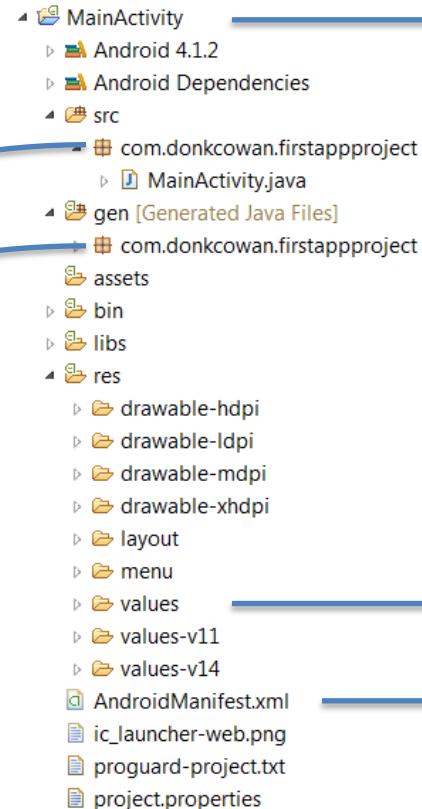
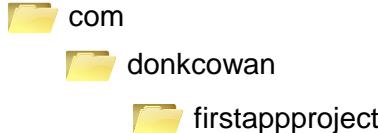
Select First App Project

Check



# Renaming App Project and Package

Actual folder structure under src and gen:



Rename Project to AddActivityExample

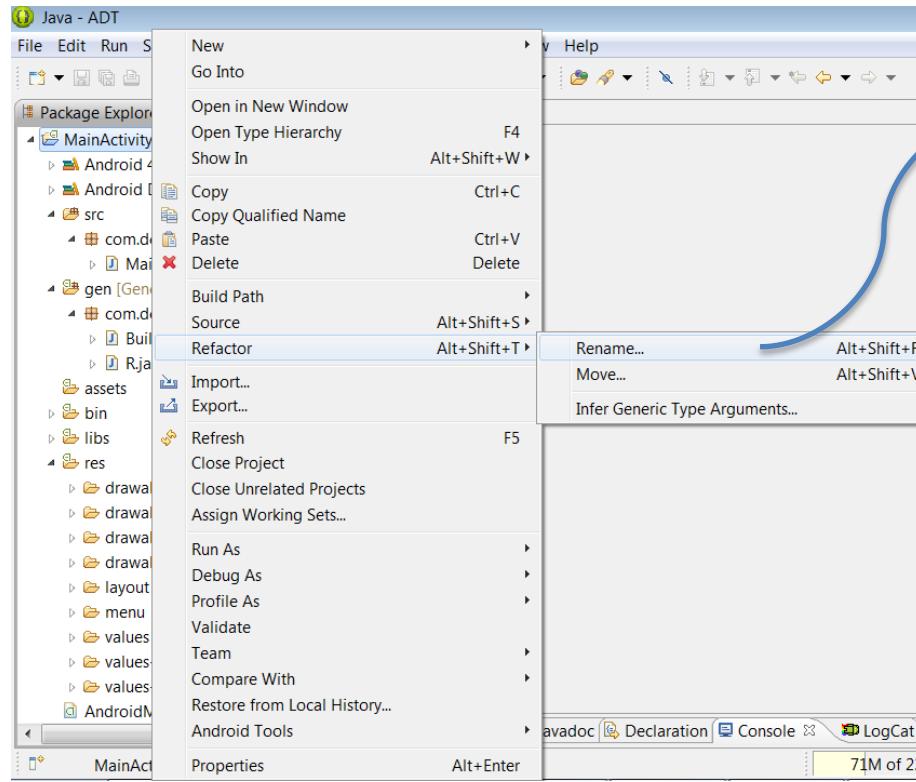
Rename Package to com.donkcowan.addactivityexample

You can use your own name or any other name you choose for the middle qualifier – or leave it as donkcowan for now. When you publish apps to Google Play, you'll need a unique qualifier, usually a domain name which assures uniqueness.

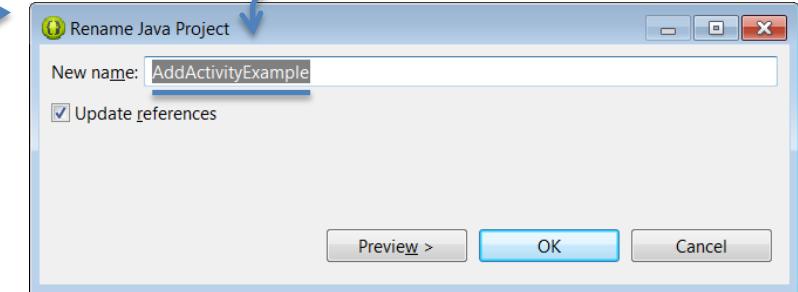
Change app\_name in string values

Verify changes in Manifest

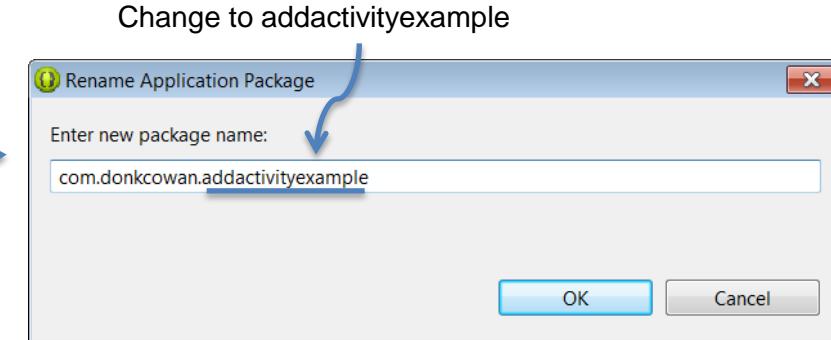
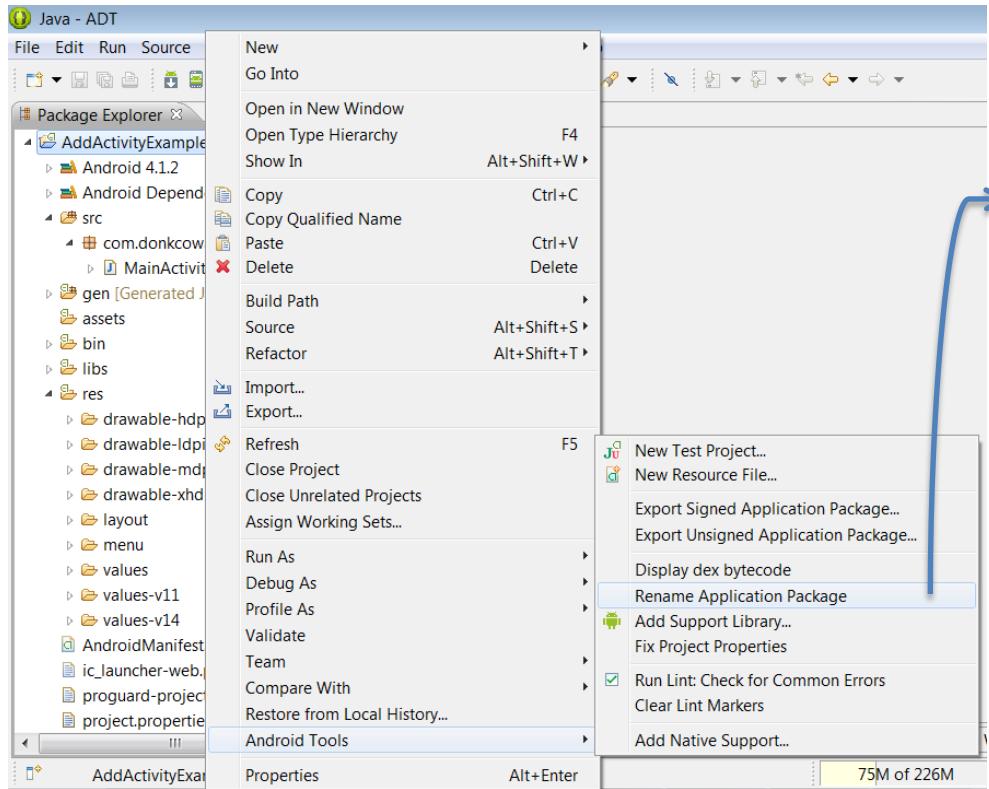
# Renaming the Project



Change to AddActivityExample



# Renaming the Application Package



Enter new package name:

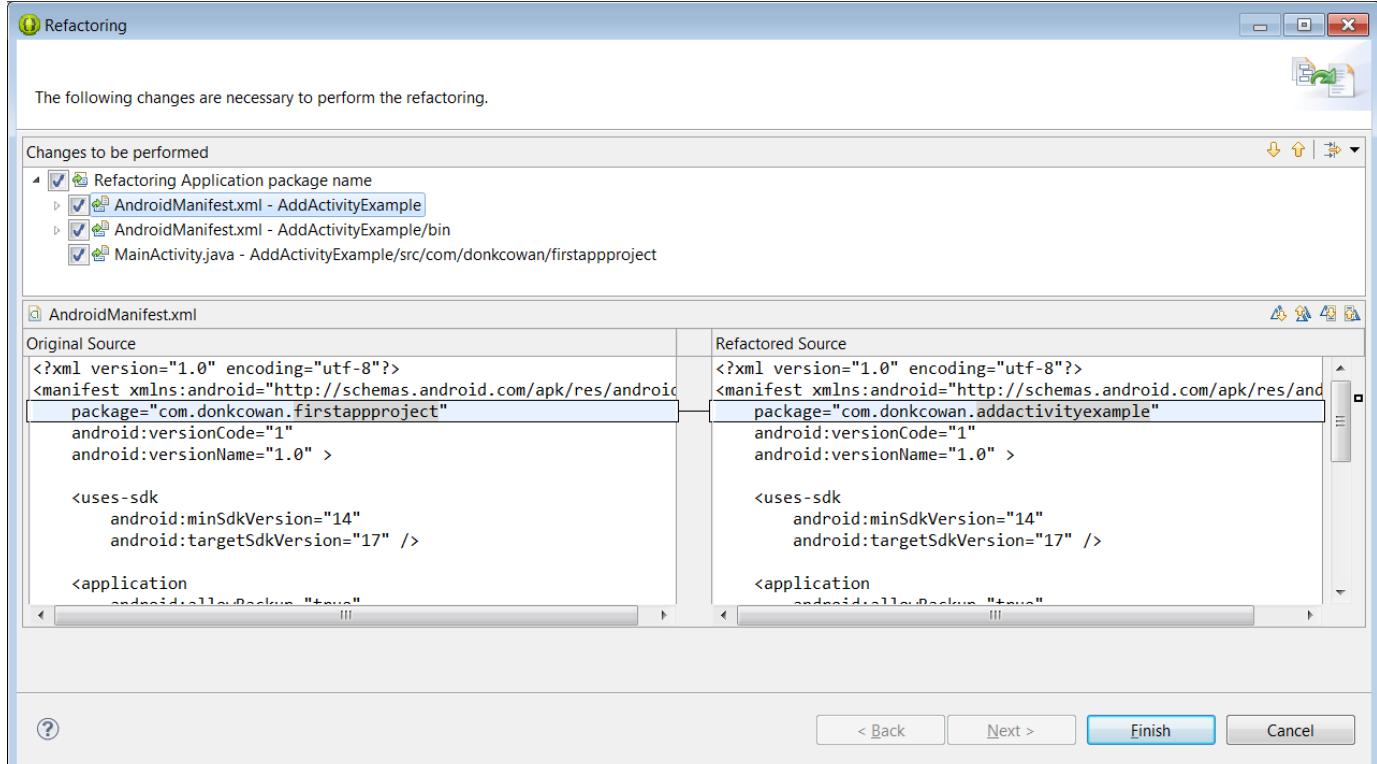
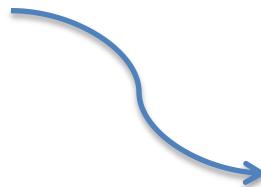
com.donkcowan.addactivityexample

OK

Cancel

# Change Verification

Gives you an opportunity to preview the changes that will be made to your application.



The following changes are necessary to perform the refactoring.

Changes to be performed

- Refactoring Application package name
  - AndroidManifest.xml - AddActivityExample
  - AndroidManifest.xml - AddActivityExample/bin
  - MainActivity.java - AddActivityExample/src/com/donkcowan/firstappproject

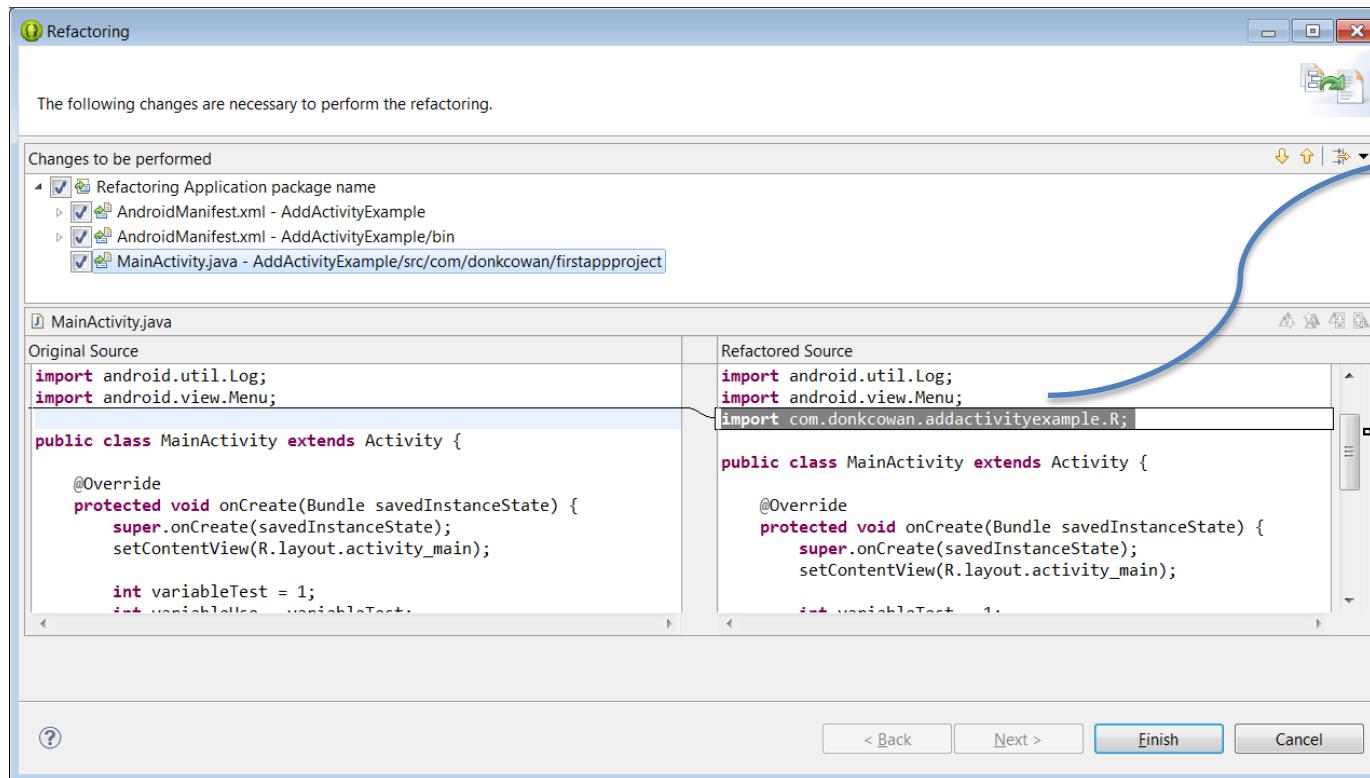
AndroidManifest.xml

Original Source	Refactored Source
<?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.donkcowan.firstappproject" android:versionCode="1" android:versionName="1.0" >  <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="17" />  <application android:allowBackup="true" ...>	<?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.donkcowan.addactivityexample" android:versionCode="1" android:versionName="1.0" >  <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="17" />  <application android:allowBackup="true" ...>

?

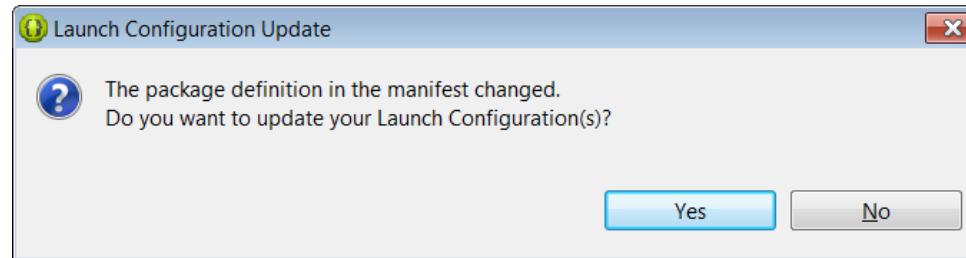
< Back    Next >    **Finish**    Cancel

# Added Import Statement

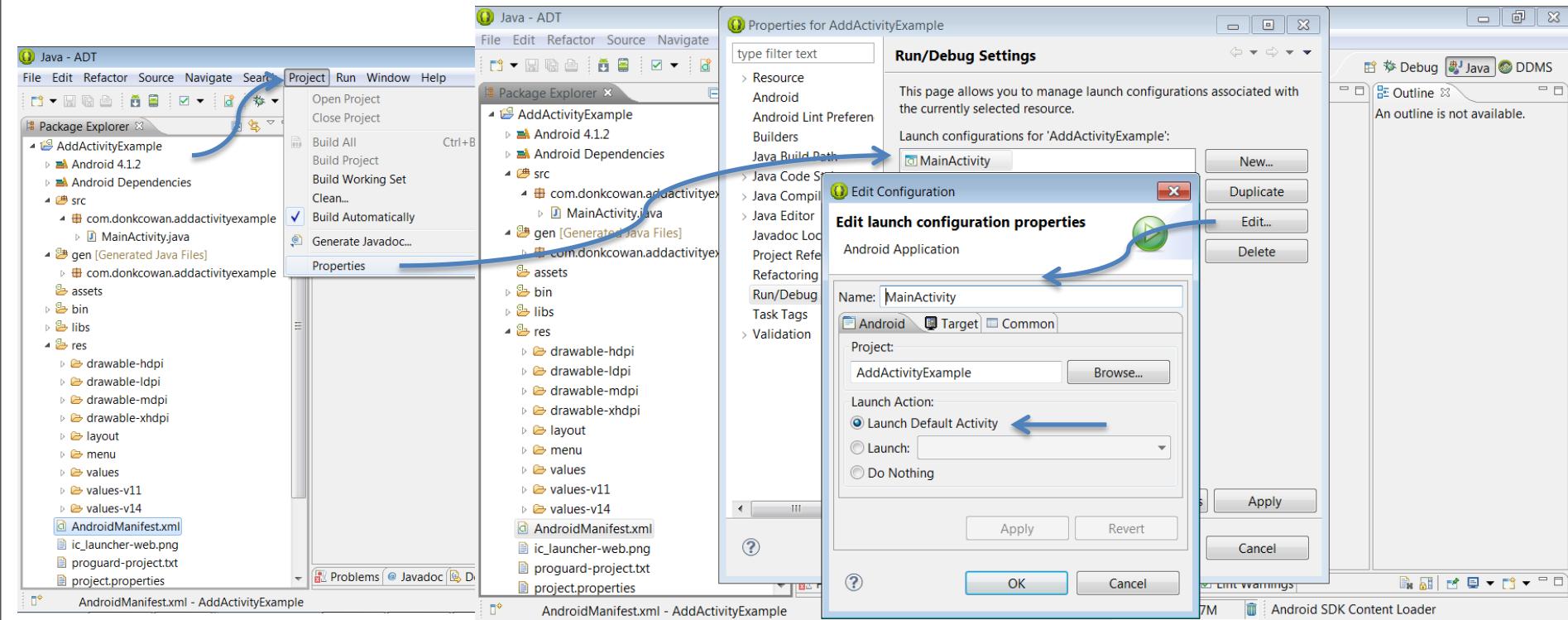


An import statement may be added to your Java source for the .R file. Once you've done all your renaming you won't need this, but it does no harm to include it.

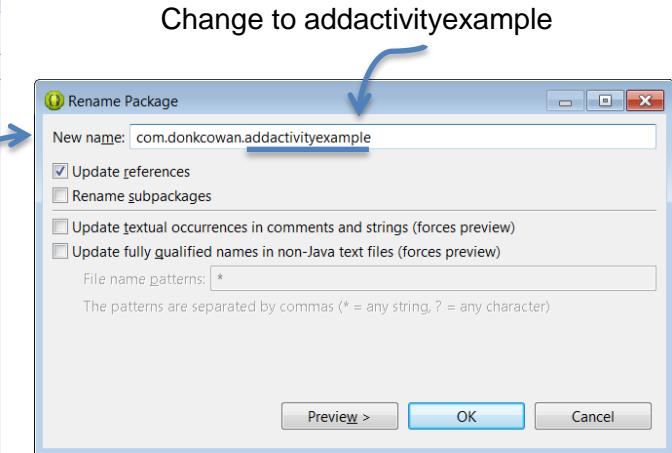
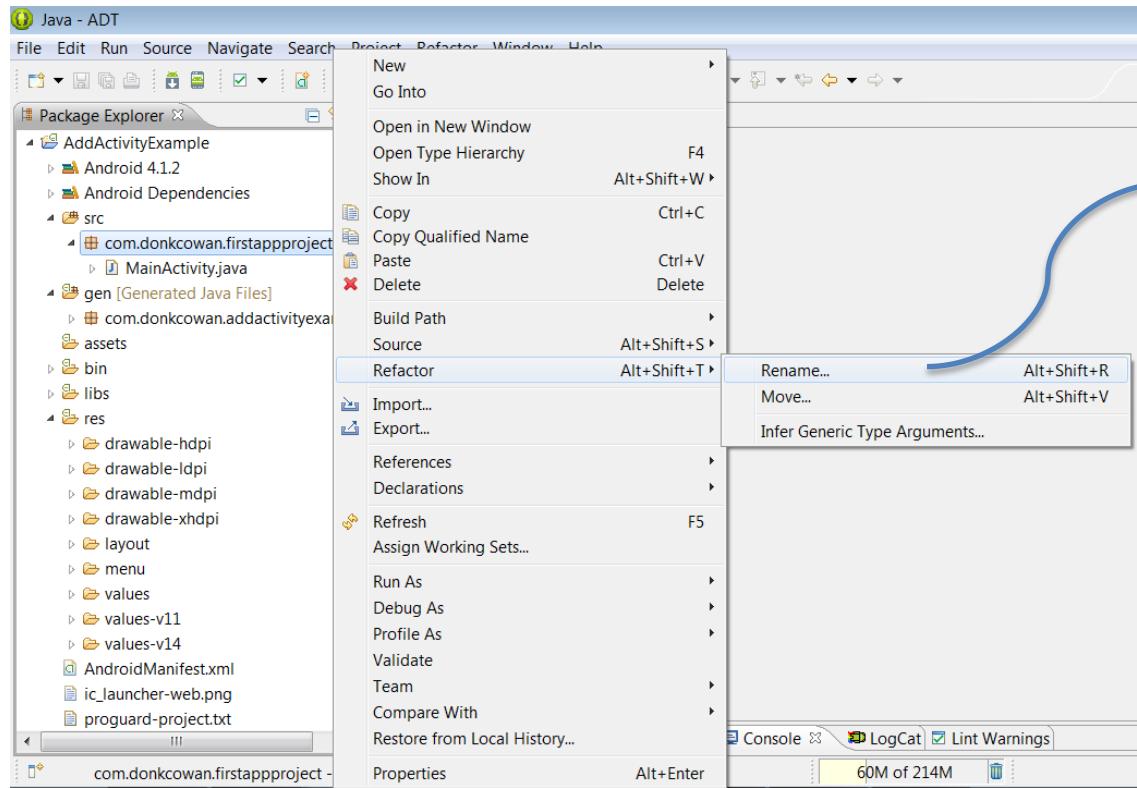
# Updating the Launch Configuration



# Launch Configuration Management

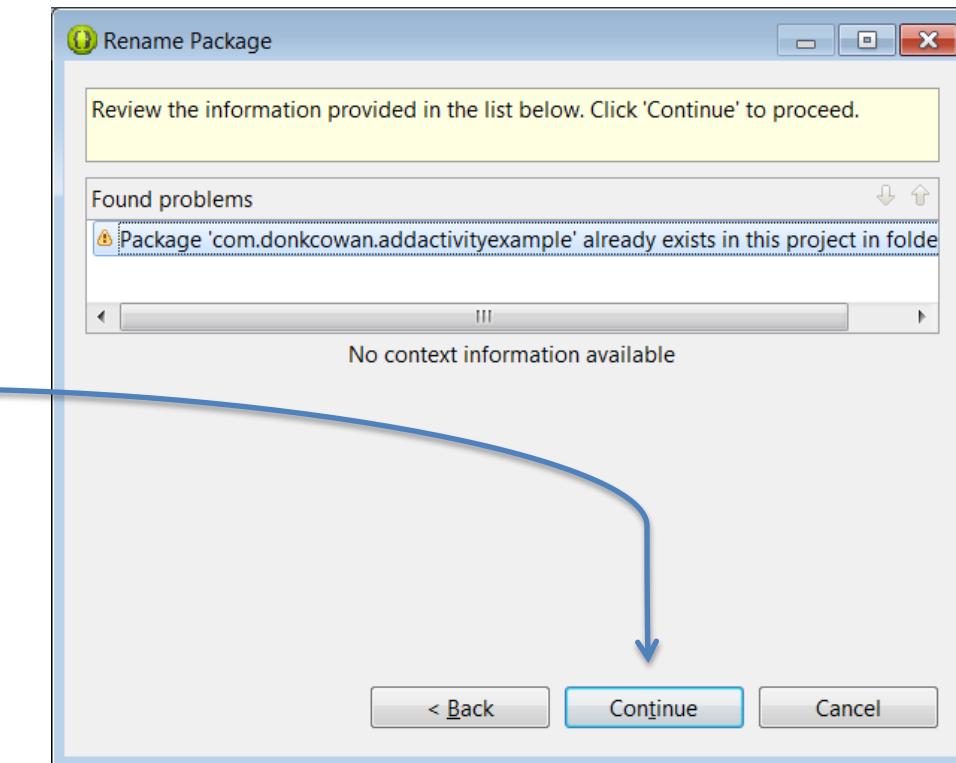


# Renaming the Src Package Name



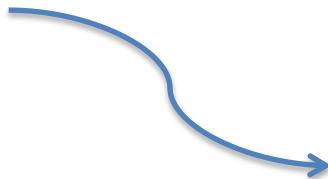
# Package Already Exists

Because you've already made changes using the addactivityexample package name, this warning appears. Just ignore it and click continue.



# Preview of Changes

Gives you an opportunity to preview the changes that will be made to your application.



**Rename Package**

Changes to be performed

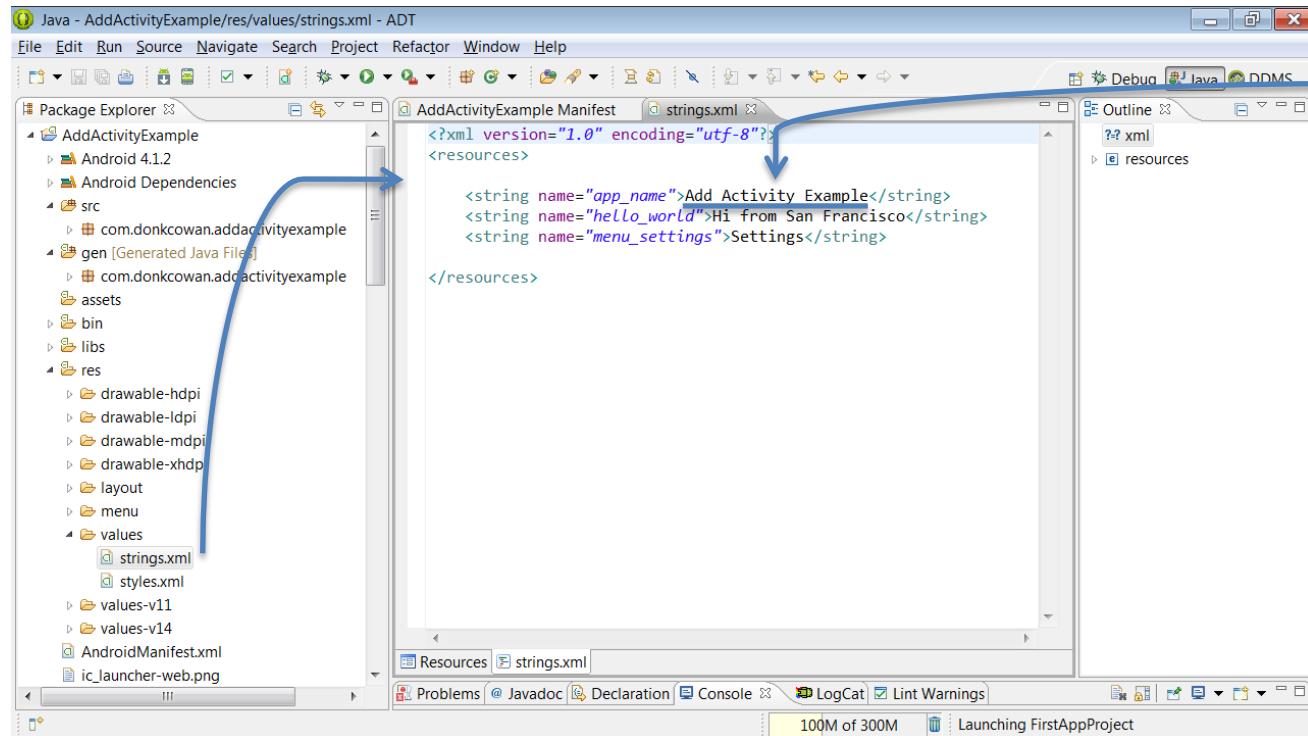
- AndroidManifest.xml - AddActivityExample/bin
- AndroidManifest.xml - AddActivityExample
- Rename package 'com.donkcowan.firstappproject' and subpackages to 'com.donkcowan.addactivityexample'
- Android Package Rename
- Update main type of launch configuration "MainActivity"

**AndroidManifest.xml**

Original Source	Refactored Source
<pre>        android:label="@string/app_name"         android:theme="@style/AppTheme" &gt;         &lt;activity             android:name="com.donkcowan.firstappproject.MainActivity"             android:label="@string/app_name" &gt;             &lt;intent-filter&gt;                 &lt;action android:name="android.intent.action.MAIN" /&gt;</pre>	<pre>        android:label="@string/app_name"         android:theme="@style/AppTheme" &gt;         &lt;activity             android:name="com.donkcowan.addactivityexample.MainActivity"             android:label="@string/app_name" &gt;             &lt;intent-filter&gt;                 &lt;action android:name="android.intent.action.MAIN" /&gt;</pre>
<pre>            &lt;category android:name="android.intent.category.LAUNCHER" /&gt;         &lt;/intent-filter&gt;     &lt;/activity&gt; &lt;/application&gt;</pre>	<pre>            &lt;category android:name="android.intent.category.LAUNCHER" /&gt;         &lt;/intent-filter&gt;     &lt;/activity&gt; &lt;/application&gt;</pre>
<pre>&lt;/manifest&gt;</pre>	<pre>&lt;/manifest&gt;</pre>

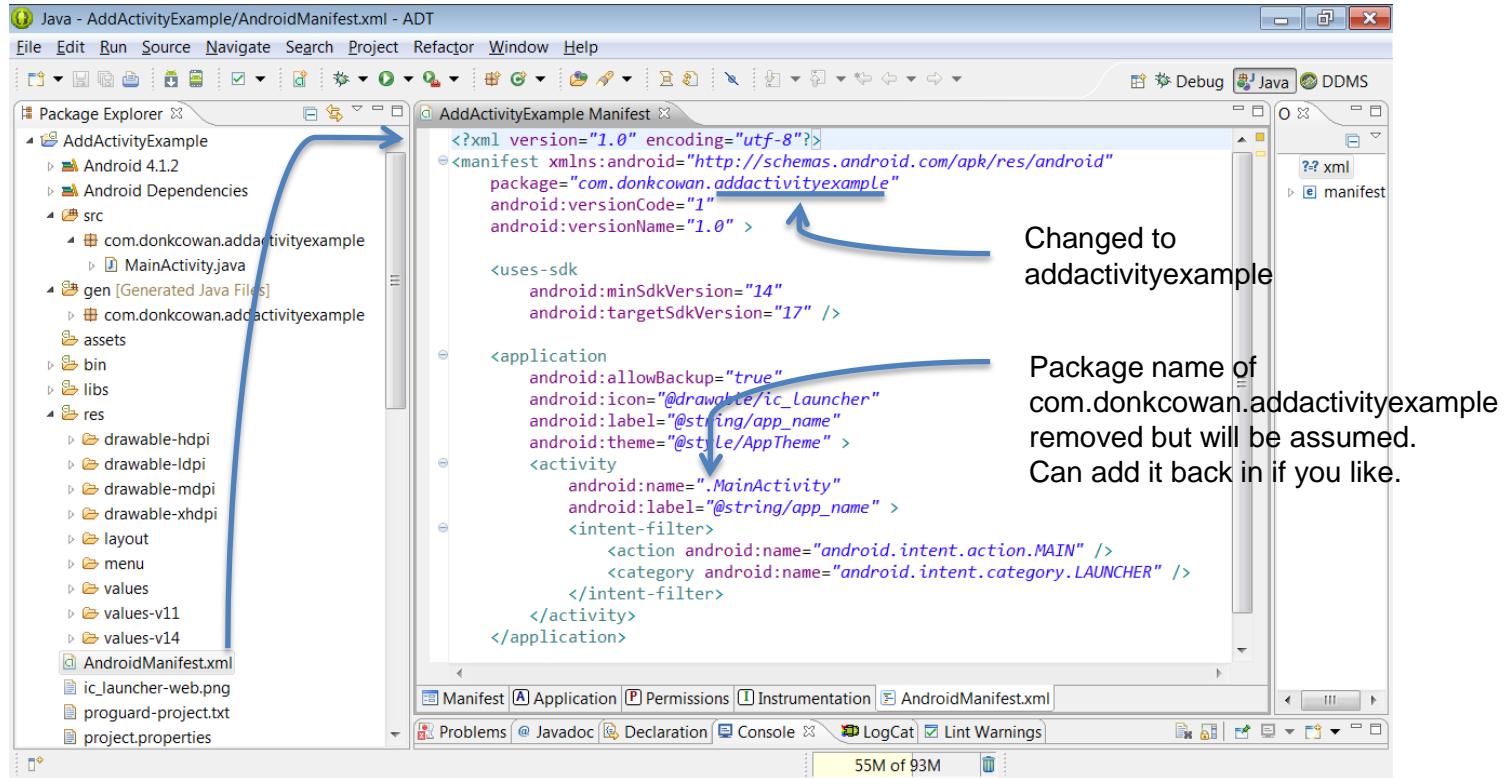
**Buttons:** < Back OK Cancel

# Changing the App Name String



Change to Add Activity Example

# Changes in the Manifest



Java - AddActivityExample/AndroidManifest.xml - ADT

File Edit Run Source Navigate Search Project Refactor Window Help

Package Explorer

- AddActivityExample
  - Android 4.1.2
  - Android Dependencies
  - src
    - com.donkcowan.addactivityexample
      - MainActivity.java
    - gen [Generated Java Files]
    - assets
    - bin
    - libs
    - res
      - drawable-hdpi
      - drawable-ldpi
      - drawable-mdpi
      - drawable-xhdpi
      - layout
      - menu
      - values
      - values-v11
      - values-v14
  - AndroidManifest.xml
  - ic\_launcher-web.png
  - proguard-project.txt
  - project.properties

AddActivityExample Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.donkcowan.addactivityexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="17" />

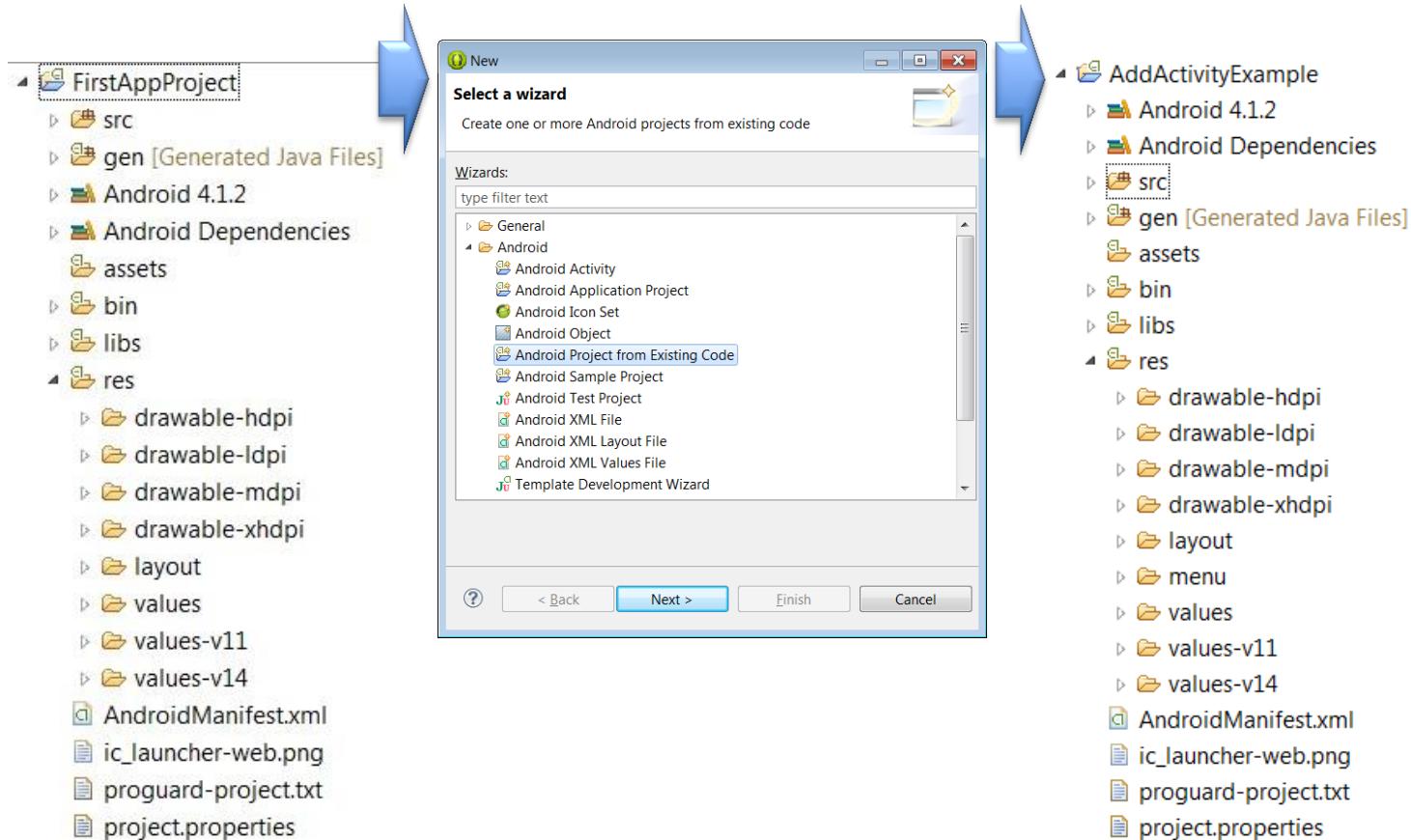
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

Manifest Application Permissions Instrumentation AndroidManifest.xml Problems Javadoc Declaration Console LogCat Lint Warnings

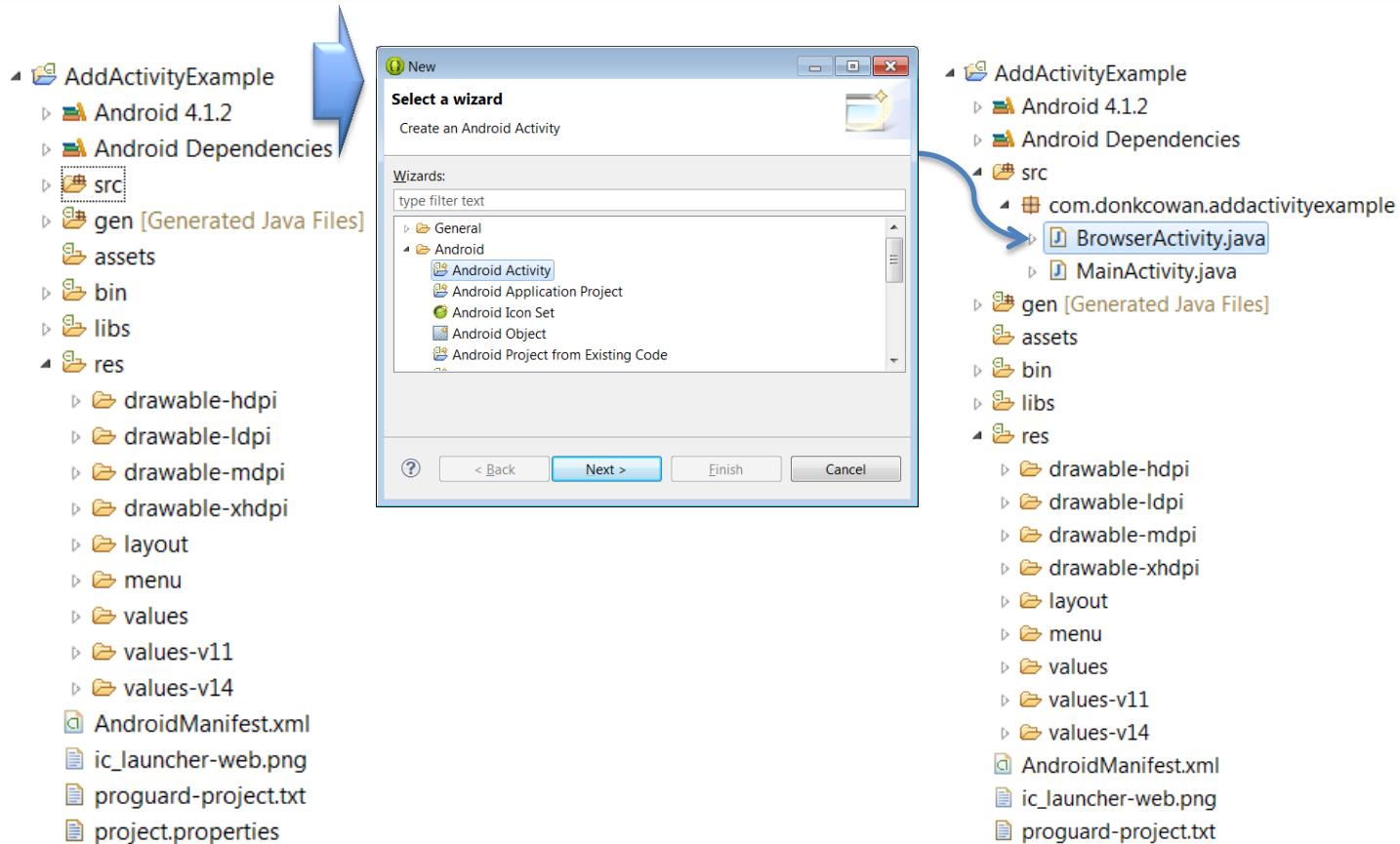
Changed to addactivityexample

Package name of com.donkcowan.addactivityexample removed but will be assumed. Can add it back in if you like.

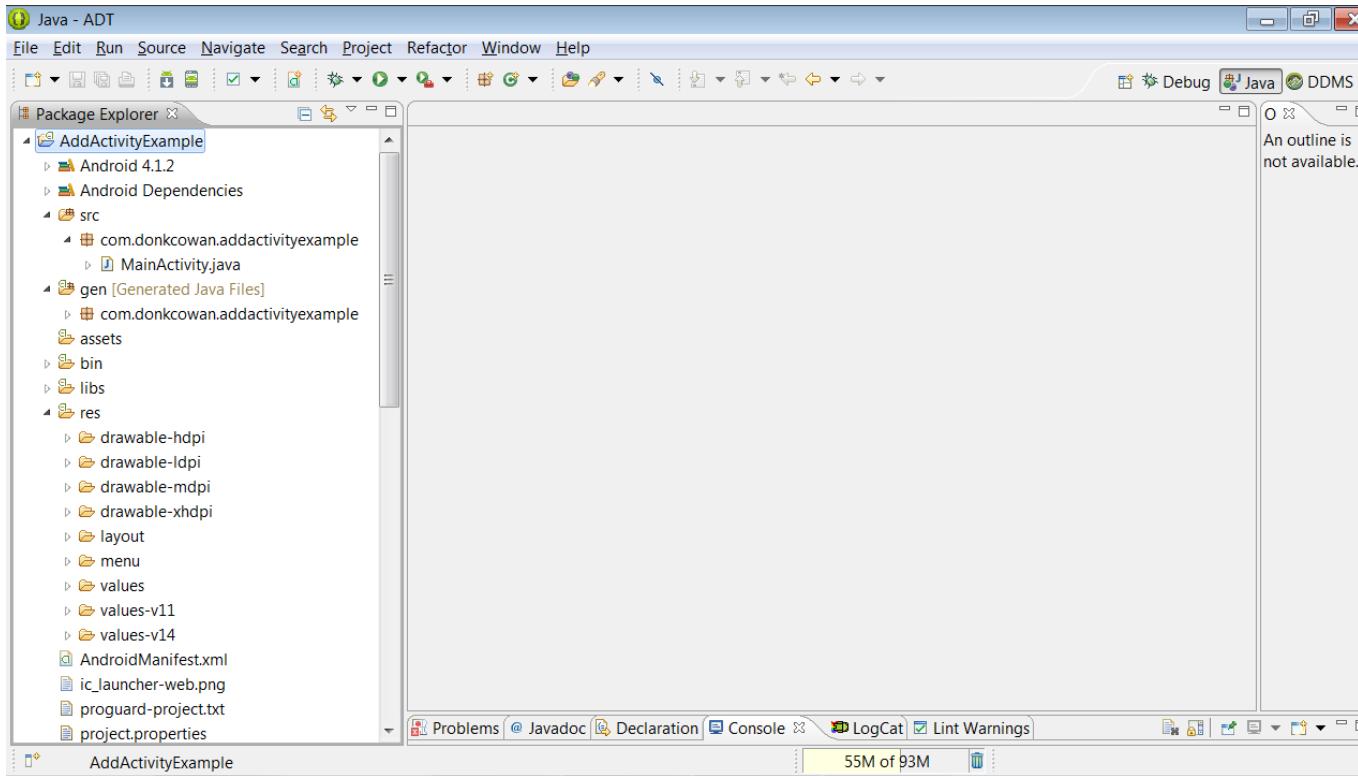
# Creating a Project from Existing Code



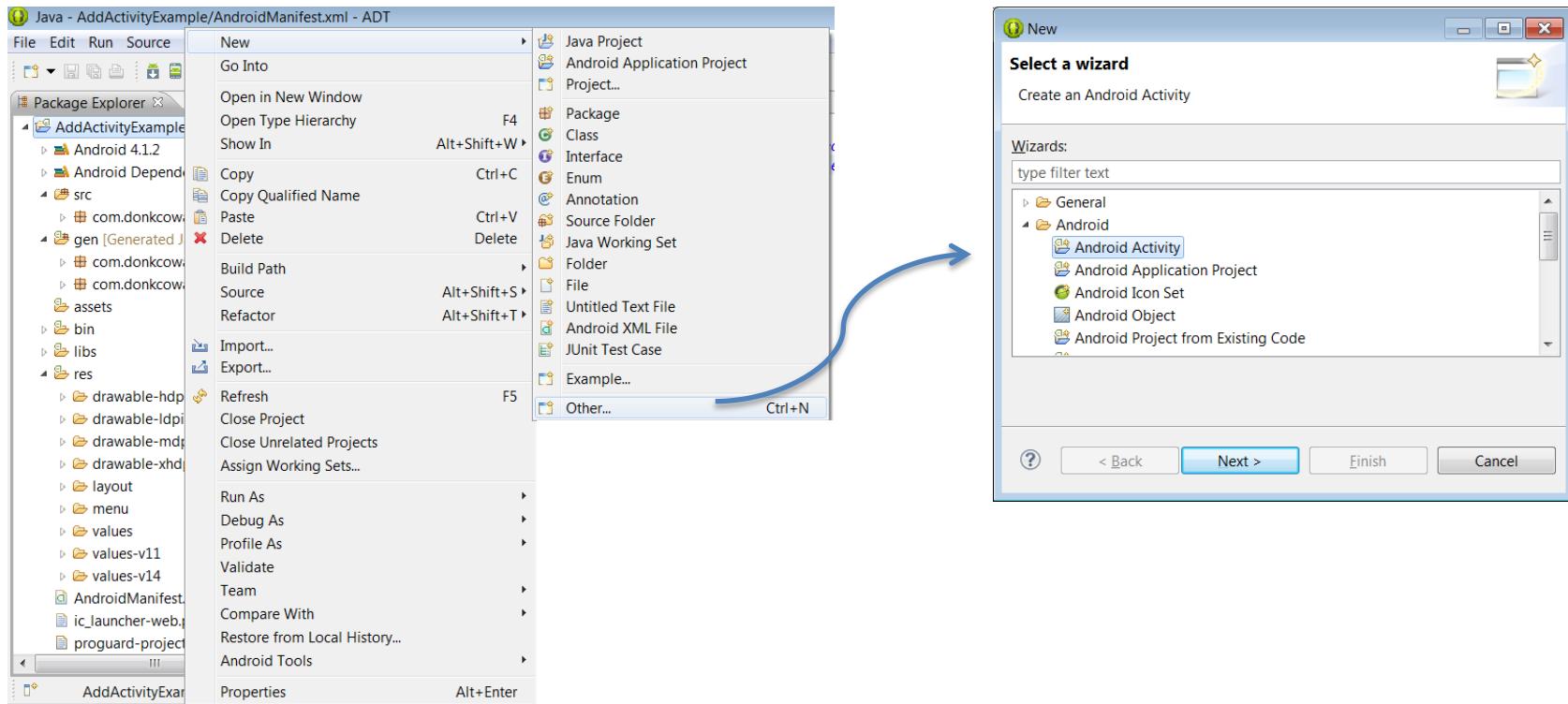
# Adding an Activity to an Existing Project



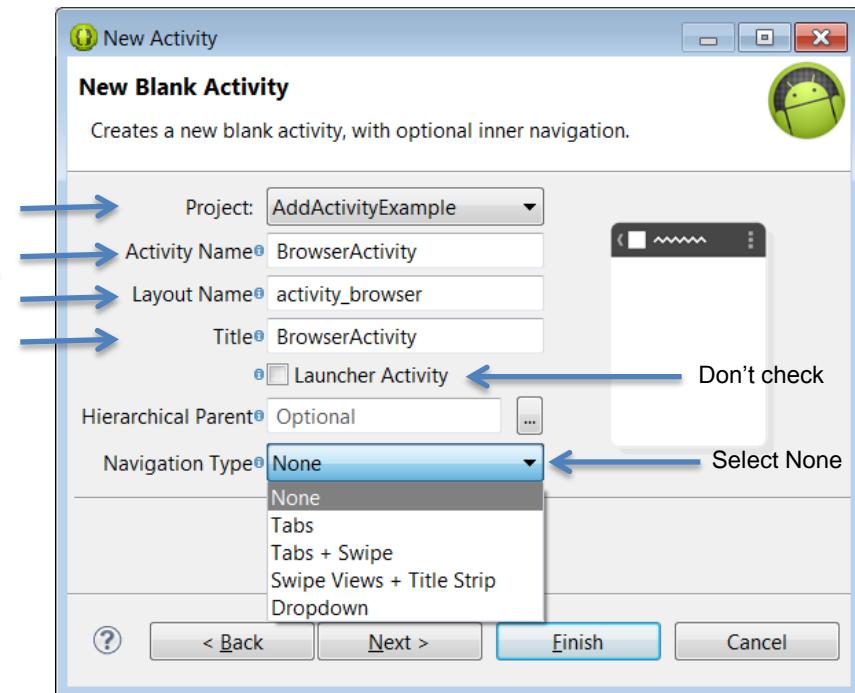
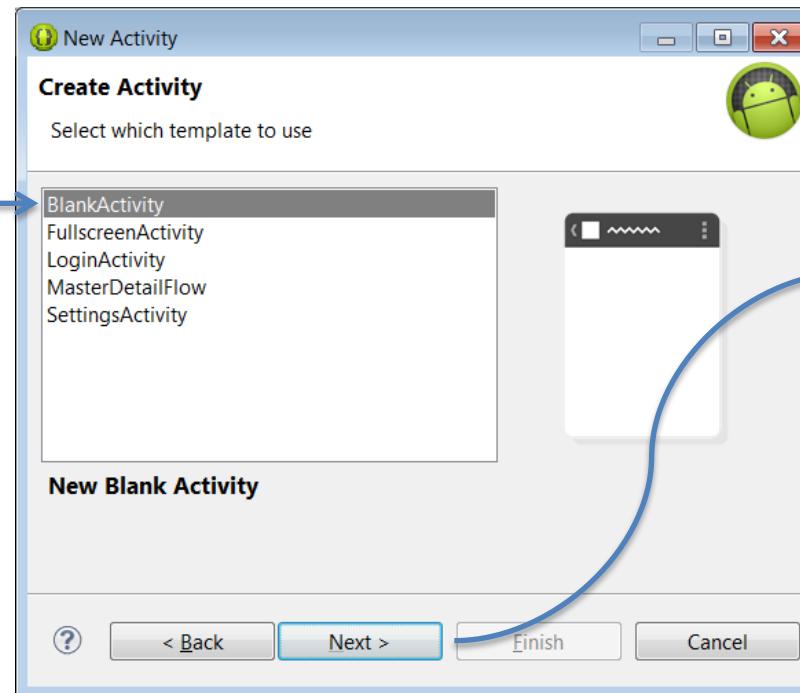
# Opening Your Workspace



# Selecting the Activity Wizard



# Naming a New Activity



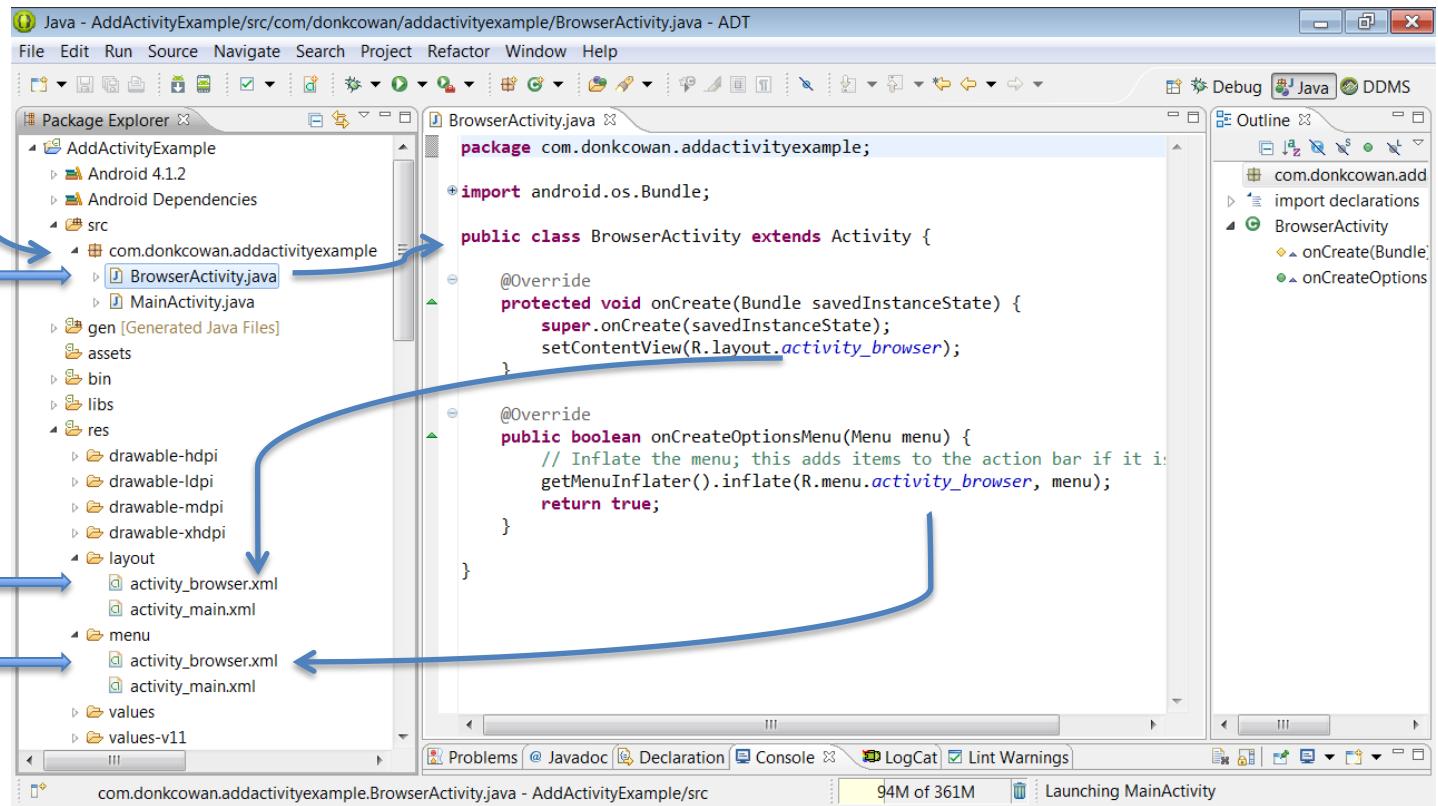
# New Activity File Additions

If source is placed in a different project folder, move it to the addactivityexample folder.

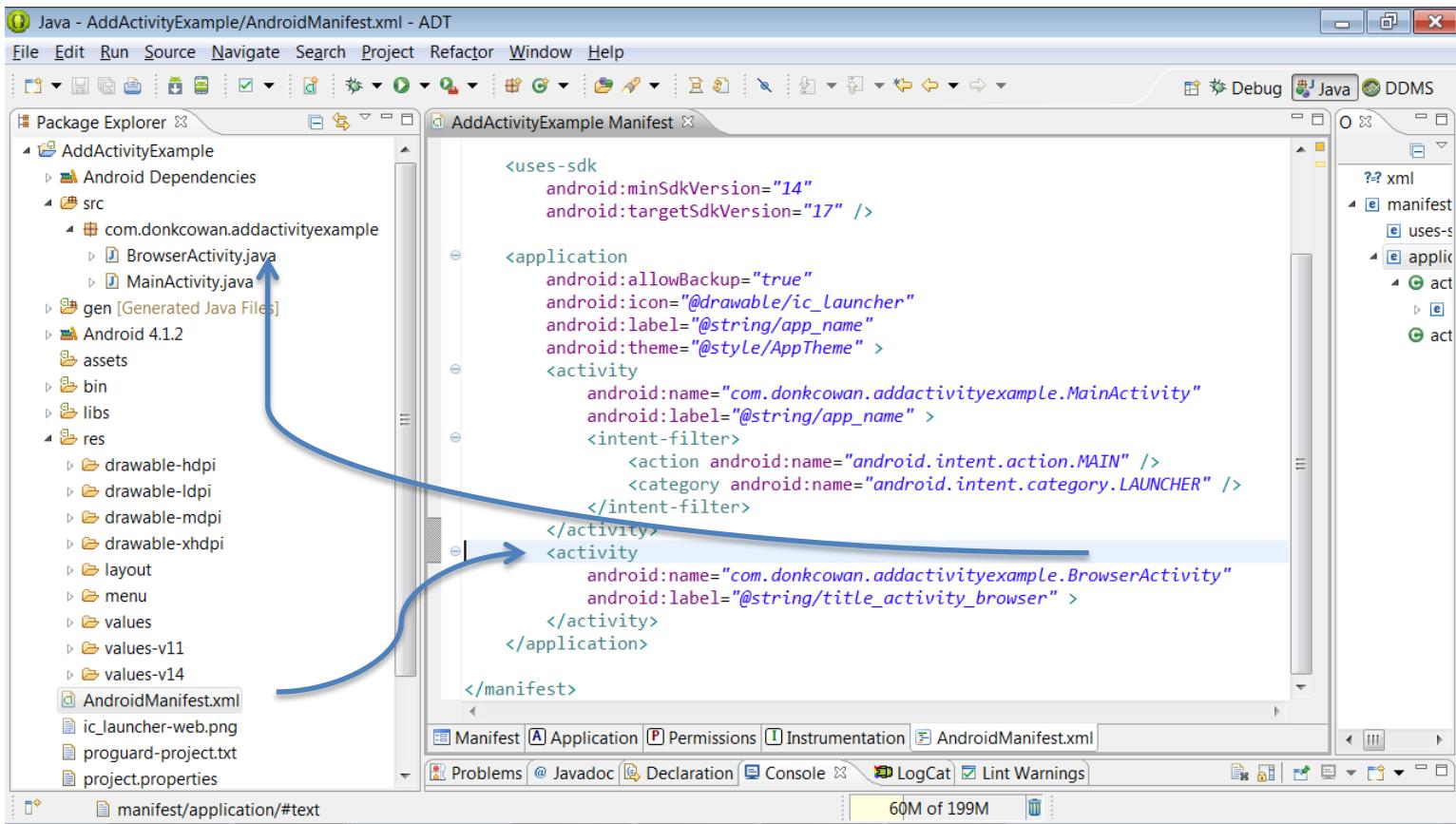
Source

Layout

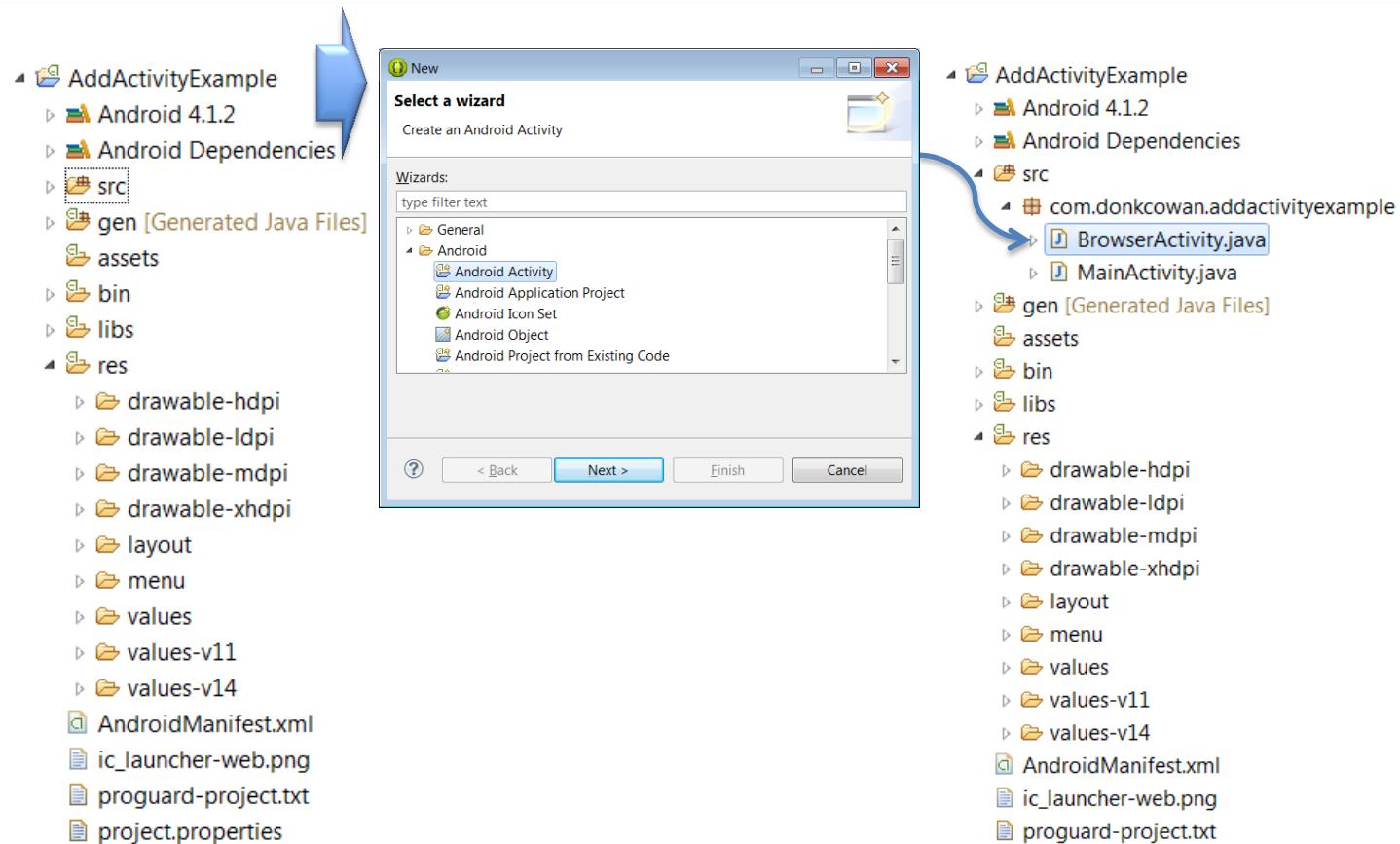
Menu



# New Activity Manifest Code

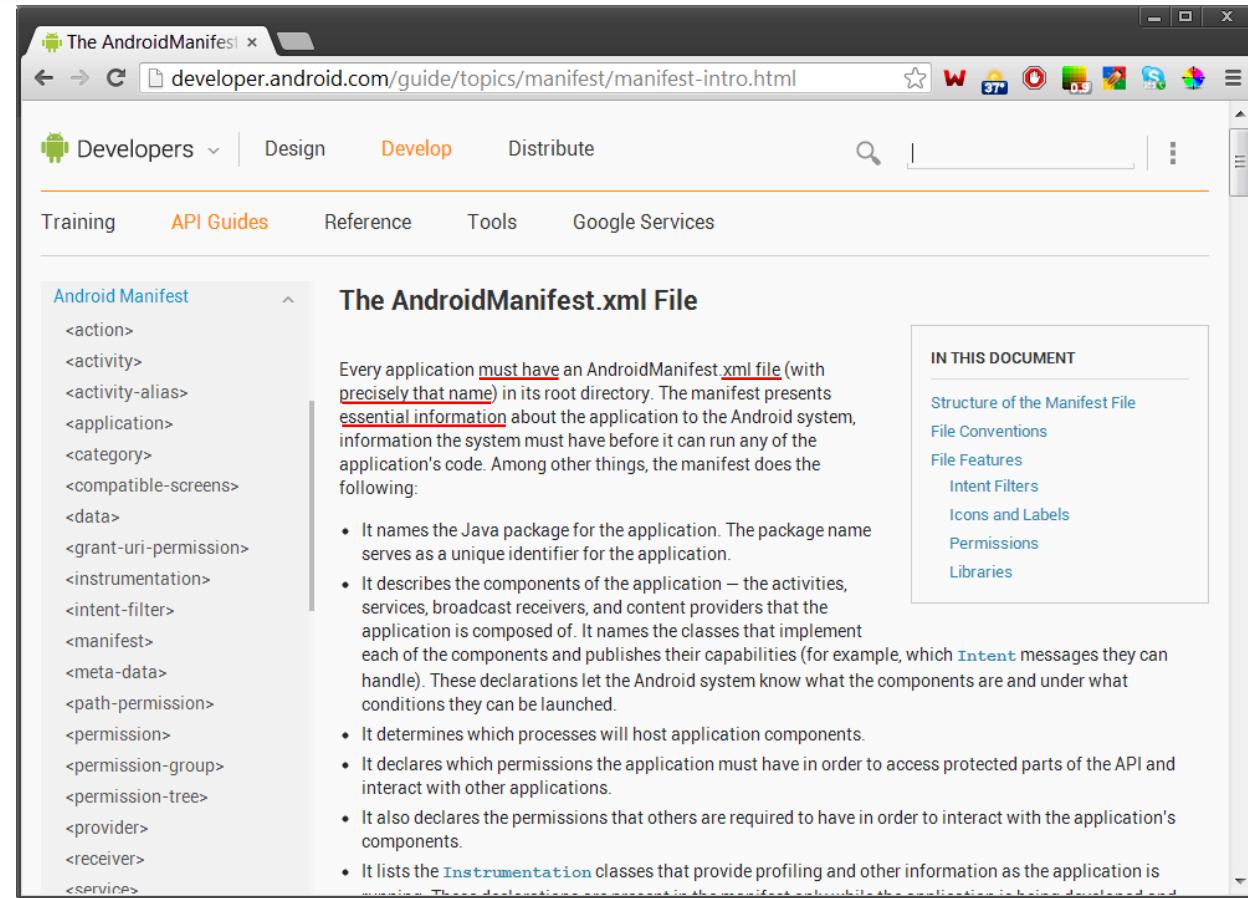


# Adding an Activity to an Existing Project



# The Android Manifest

Portions of this page are reproduced from work created and shared by the [Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).

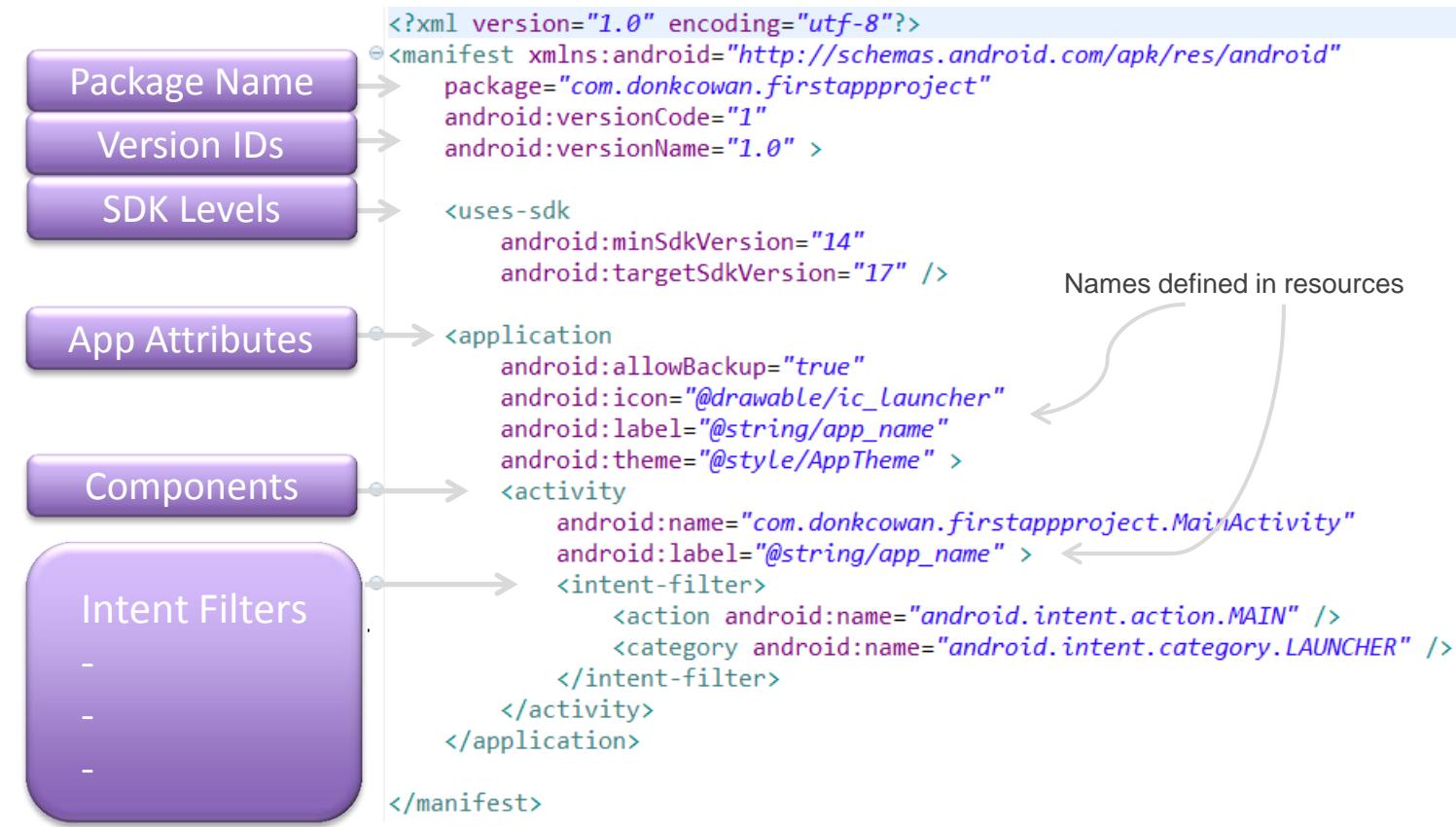


Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. Among other things, the manifest does the following:

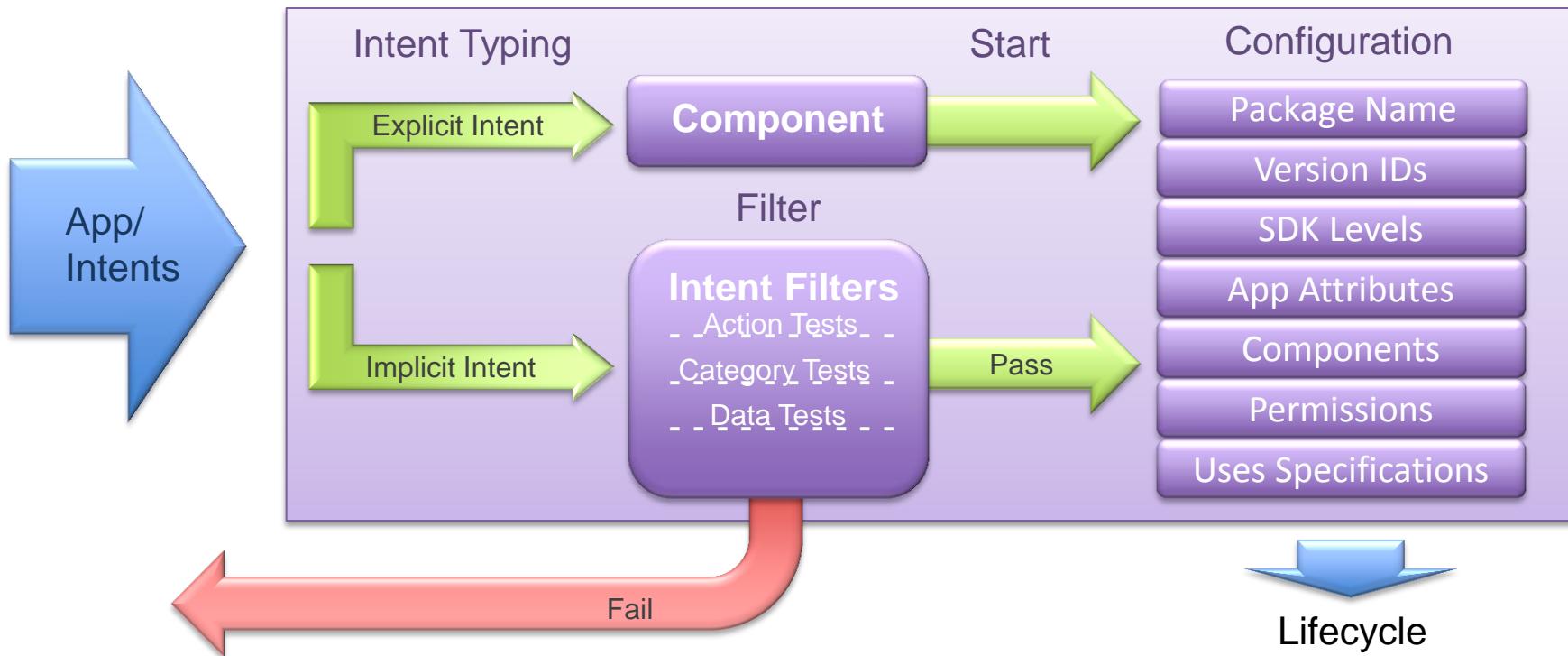
- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application – the activities, services, broadcast receivers, and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities (for example, which Intent messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the `Instrumentation` classes that provide profiling and other information as the application is running. These declarations are particularly important for instrumentation testing.

# First App Manifest

```
FirstAppProject
  - src
  - gen [Generated Java Files]
  - Android 4.1.2
  - Android Dependencies
    - assets
  - bin
  - libs
  - res
    - drawable-hdpi
    - drawable-ldpi
    - drawable-mdpi
    - drawable-xhdpi
    - layout
    - values
    - values-v11
    - values-v14
  - AndroidManifest.xml
  - ic_launcher-web.png
  - proguard-project.txt
  - project.properties
```



# Manifest Uses



# Intent Filter Standard Actions

Most important and frequently used

Start up component as the initial activity of a task.

Display data for the user to edit.

## Standard Activity Actions (to be performed)

ACTION\_MAIN  
ACTION\_VIEW  
ACTION\_ATTACH\_DATA  
ACTION\_EDIT  
ACTION\_PICK  
ACTION\_CHOOSER  
ACTION\_GET\_CONTENT  
ACTION\_DIAL  
ACTION\_CALL  
ACTION\_SEND  
ACTION\_SENDTO  
ACTION\_ANSWER  
ACTION\_INSERT  
ACTION\_DELETE  
ACTION\_RUN  
ACTION\_SYNC  
ACTION\_PICK\_ACTIVITY  
ACTION\_SEARCH  
ACTION\_WEB\_SEARCH  
ACTION\_FACTORY\_TEST

## Standard Broadcast Actions (were performed)

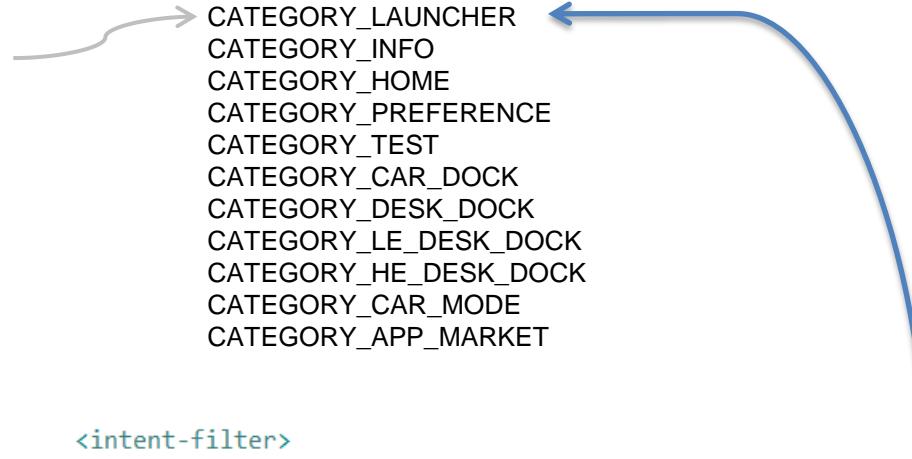
ACTION\_TIME\_TICK  
ACTION\_TIME\_CHANGED  
ACTION\_TIMEZONE\_CHANGED  
ACTION\_BOOT\_COMPLETED  
ACTION\_PACKAGE\_ADDED  
ACTION\_PACKAGE\_CHANGED  
ACTION\_PACKAGE\_REMOVED  
ACTION\_PACKAGE\_RESTARTED  
ACTION\_PACKAGE\_DATA\_CLEARED  
ACTION\_UID\_REMOVED  
ACTION\_BATTERY\_CHANGED  
ACTION\_POWER\_CONNECTED  
ACTION\_POWER\_DISCONNECTED  
ACTION\_SHUTDOWN

First App:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

# Intent Filter Standard Categories

The activity can be the initial activity of a task and is listed in the top-level application launcher.



- CATEGORY\_DEFAULT
- CATEGORY\_BROWSABLE
- CATEGORY\_TAB
- CATEGORY\_ALTERNATIVE
- CATEGORY\_SELECTED\_ALTERNATIVE
- CATEGORY\_LAUNCHER**
- CATEGORY\_INFO
- CATEGORY\_HOME
- CATEGORY\_PREFERENCE
- CATEGORY\_TEST
- CATEGORY\_CAR\_DOCK
- CATEGORY\_DESK\_DOCK
- CATEGORY\_LE\_DESK\_DOCK
- CATEGORY\_HE\_DESK\_DOCK
- CATEGORY\_CAR\_MODE
- CATEGORY\_APP\_MARKET

**First App:**

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

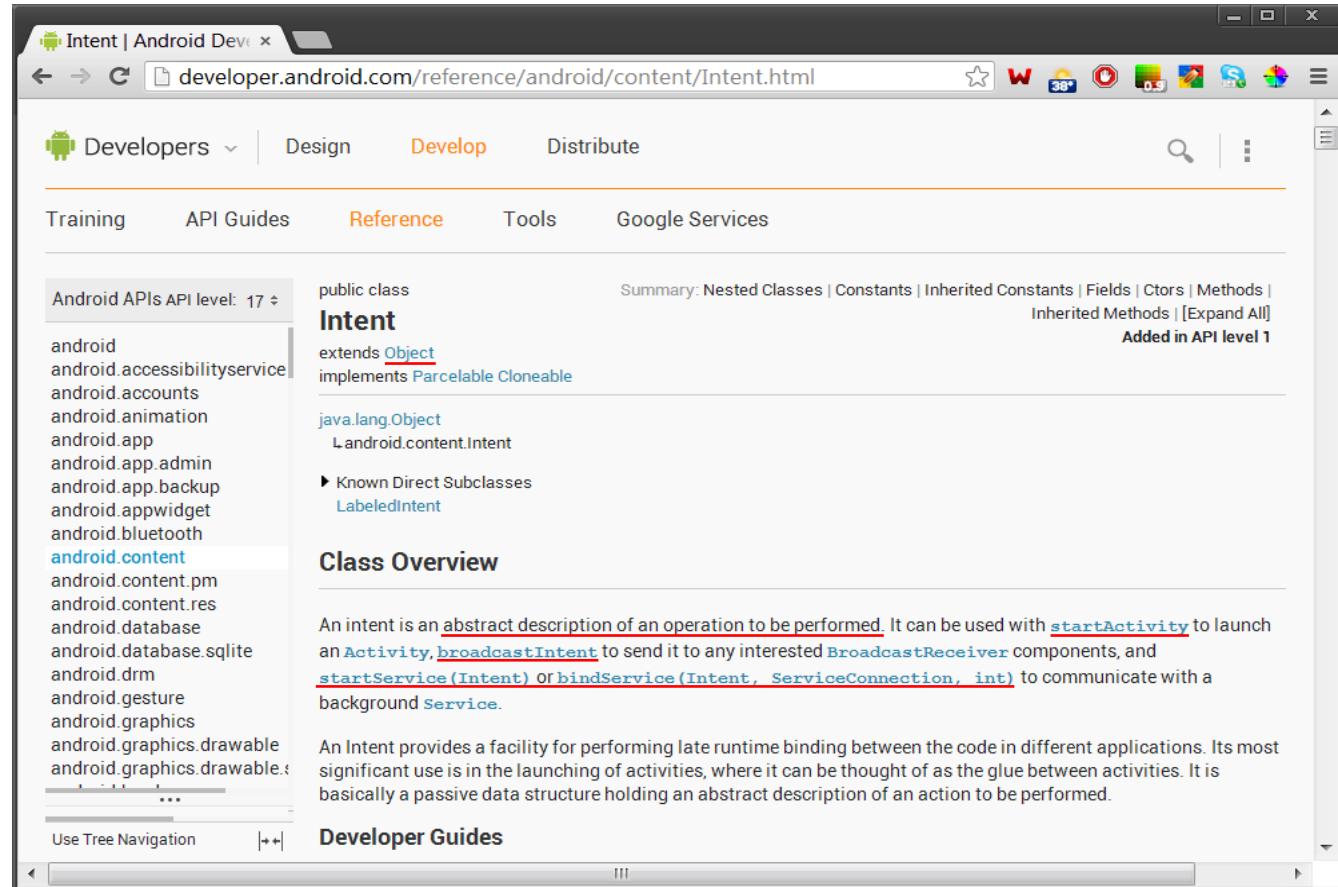
# Manifest XML Tags

[<action>](#)  
[<activity>](#)  
[<activity-alias>](#)  
[<application>](#)  
[<category>](#)  
[<data>](#)  
[<grant-uri-permission>](#)  
[<instrumentation>](#)  
[<intent-filter>](#)  
[<manifest>](#)  
[<meta-data>](#)

[<permission>](#)  
[<permission-group>](#)  
[<permission-tree>](#)  
[<provider>](#)  
[<receiver>](#)  
[<service>](#)  
[<supports-screens>](#)  
[<uses-configuration>](#)  
[<uses-feature>](#)  
[<uses-library>](#)  
[<uses-permission>](#)  
[<uses-sdk>](#)

# Intents

[http://developer.android.com  
/reference/android/content/Intent.html](http://developer.android.com/reference/android/content/Intent.html)



The screenshot shows a web browser displaying the official Android developer documentation for the `Intent` class. The URL in the address bar is <http://developer.android.com/reference/android/content/Intent.html>. The page is part of the "Develop" section of the Android Developers site.

The main content area shows the `Intent` class definition:

```
public class Intent
    extends Object
    implements Parcelable, Cloneable
```

It also lists the `java.lang.Object` interface and its subclasses `android.content.Intent` and `LabeledIntent`.

### Class Overview

An intent is an abstract description of an operation to be performed. It can be used with `startActivity` to launch an `Activity`, `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background service.

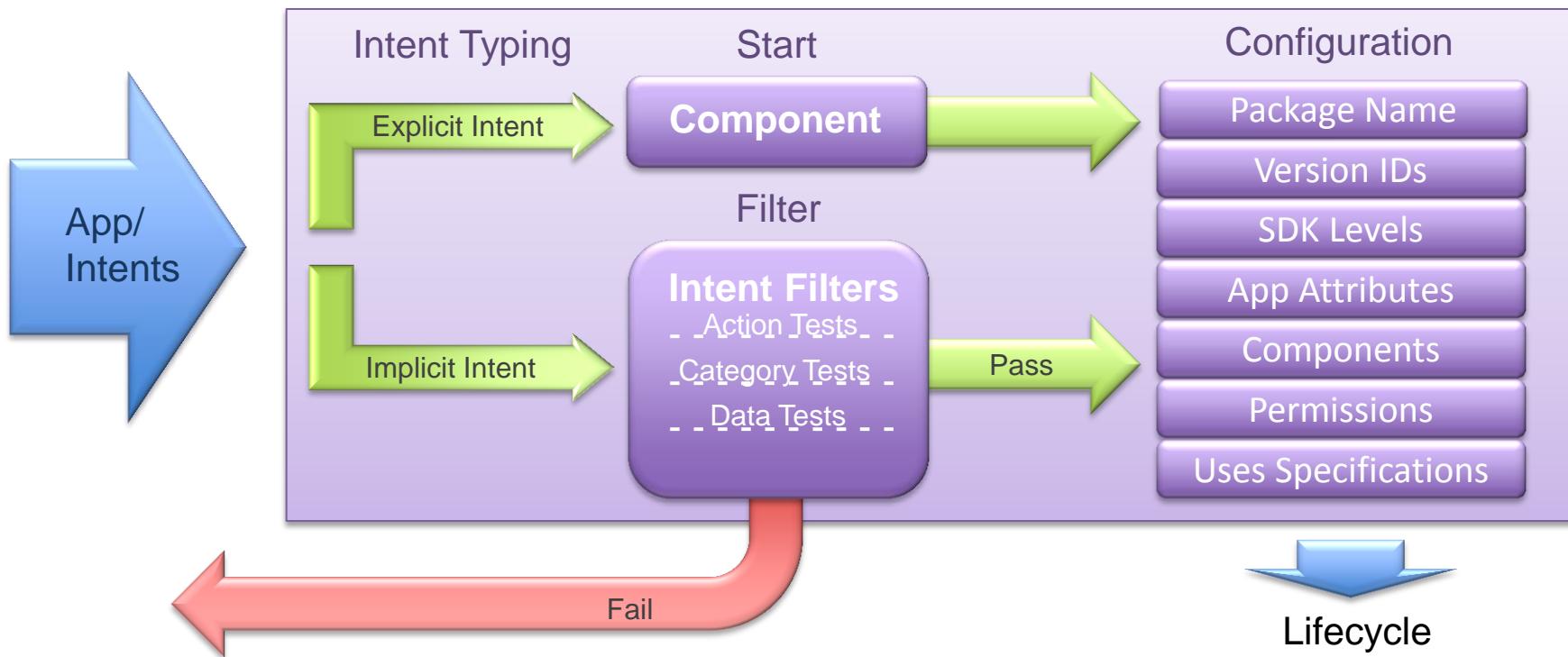
An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

On the left side, there is a sidebar navigation menu with the following items:

- Android APIs API level: 17
- android
- android.accessibilityservice
- android.accounts
- android.animation
- android.app
- android.app.admin
- android.app.backup
- android.appwidget
- android.bluetooth
- android.content**
- android.content.pm
- android.content.res
- android.database
- android.database.sqlite
- android.drm
- android.gesture
- android.graphics
- android.graphics.drawable
- ...

At the bottom of the sidebar, there are two buttons: "Use Tree Navigation" and "Developer Guides".

# Intent Typing and Filtering



# Primary Intent Information

**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](#), such as <http://www.google.com>

## Examples

**action      data**

[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\\_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.

# Other Intent Information

In addition to these primary attributes, there are a number of secondary attributes that you can also include with an intent:

**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.

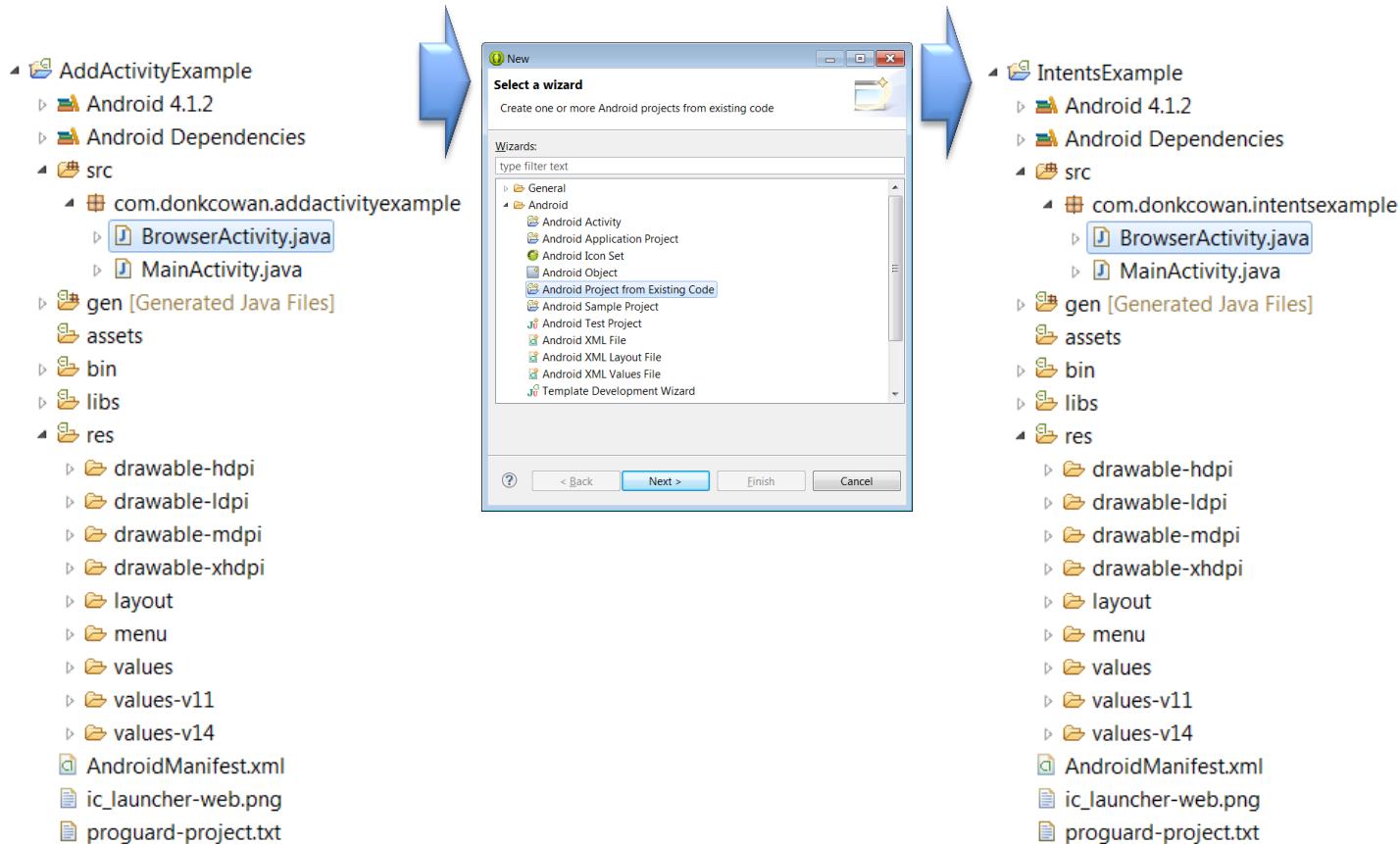
**type** -- Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.

**component** -- Specifies an explicit name of a component class to use for the intent. Normally this is determined by looking at the other information in the intent (the action, data/type, and categories) and matching that with a component that can handle it. If this attribute is set then none of the evaluation is performed, and this component is used exactly as is. By specifying this attribute, all of the other Intent attributes become optional.

**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 an action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

For more information see: <http://developer.android.com/reference/android/content/Intent.html>

# Creating a New Project to Demonstrate Intents



# Explicit Intent Example

```
package com.donkcowan.intentsexample;

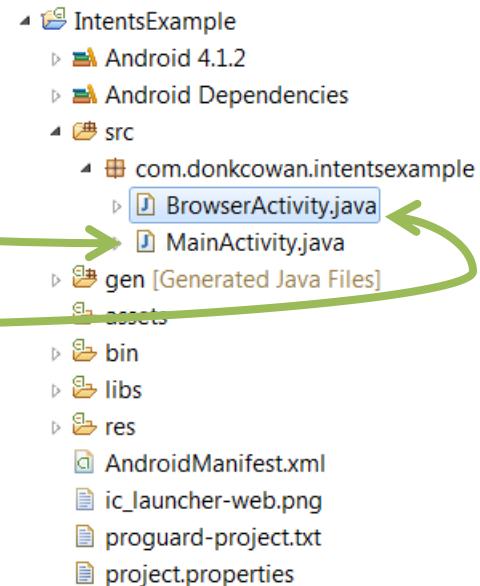
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent browserIntent = new Intent(this, BrowserActivity.class);
        startActivity(browserIntent);
    }

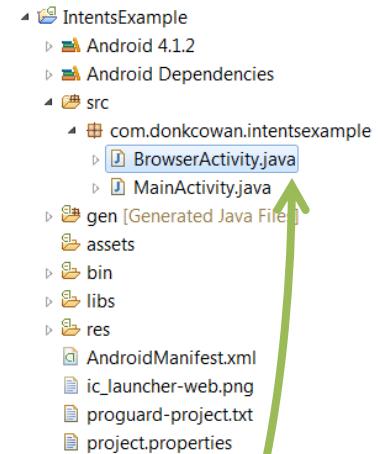
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```



# Explicit Intent for Activity Start

```
Intent browserIntent = new Intent(this, BrowserActivity.class);
startActivity(browserIntent);
```

→ Declares object as an intent  
→ name of intent object  
→ new instance of object  
→ Intent class  
→ context of **this** activity  
→ Name of component class  
→ class indication  
→ Intent for activity to be started  
→ startActivity method  
method signature to create an intent for a specific component



# Implicit Intent Example

```
package com.donkowan.intentsexample;

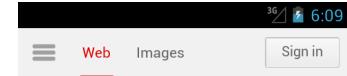
import android.net.Uri;

public class BrowserActivity extends Activity {

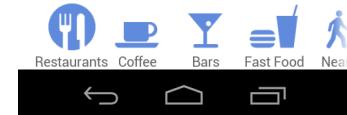
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_browser);

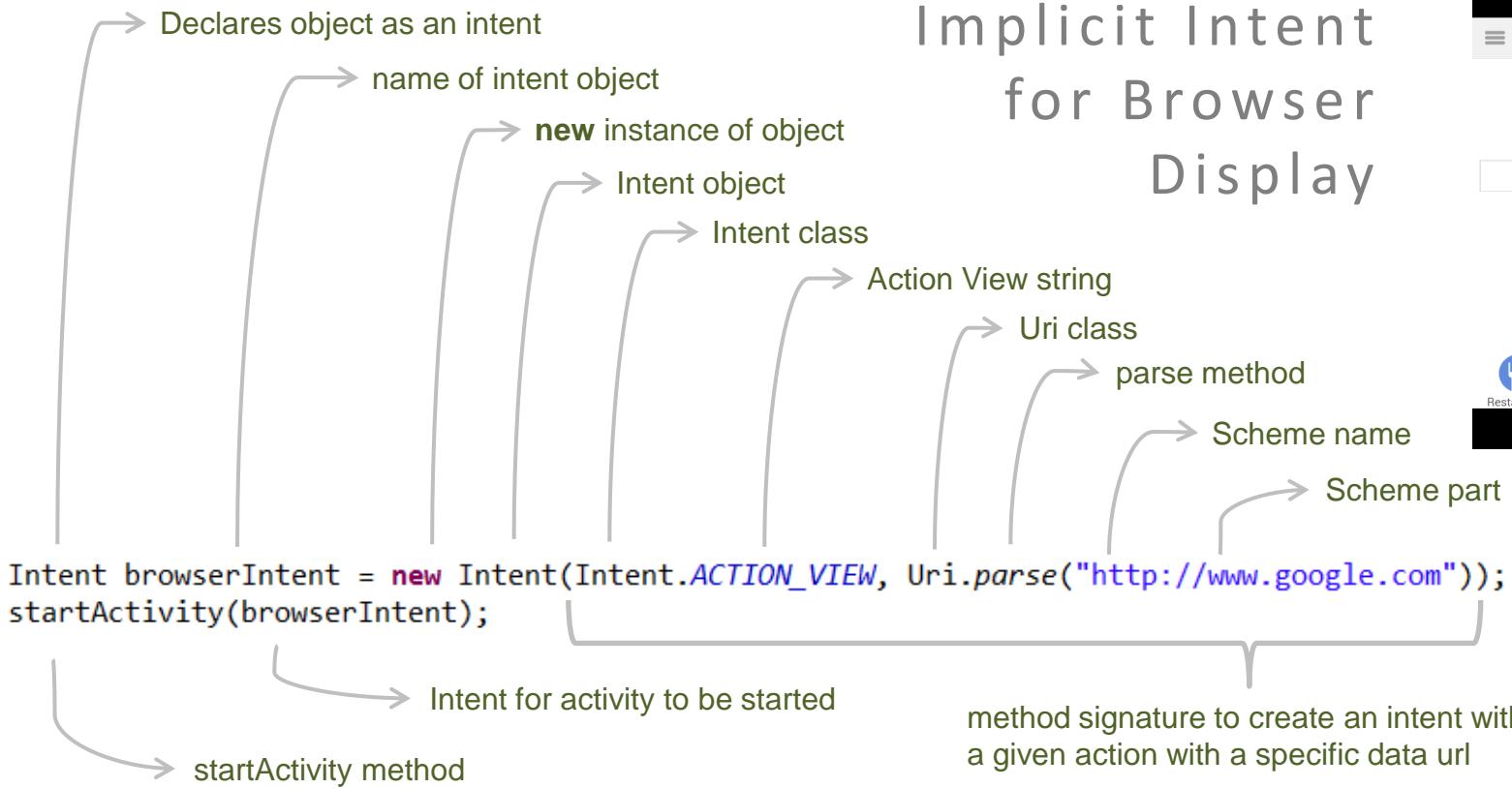
        Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
        startActivity(browserIntent);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_browser, menu);
        return true;
    }
}
```

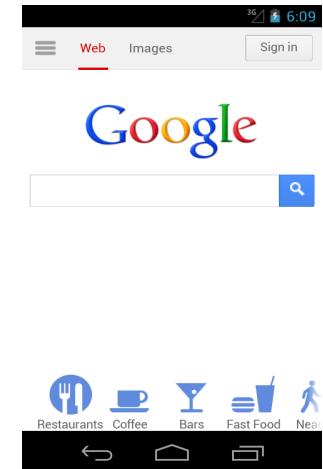


Google

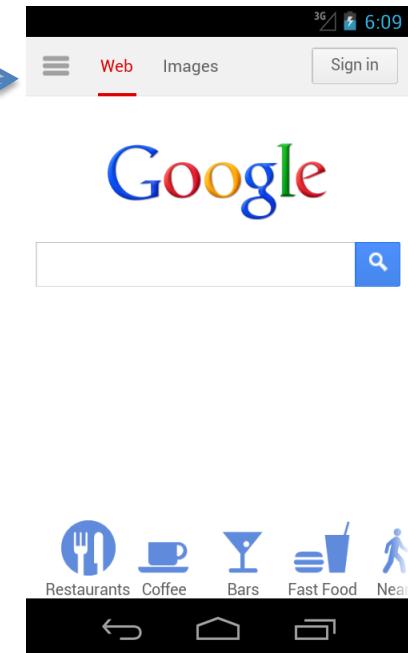
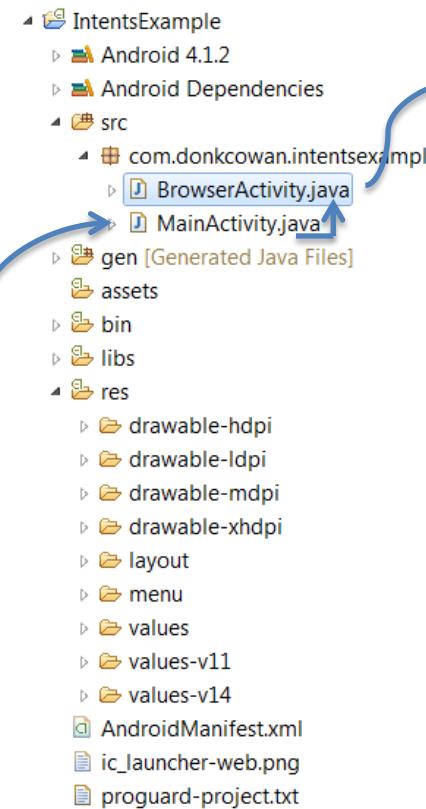




## Implicit Intent for Browser Display



# How Intents were used in Our Example



# Listener for Button Click

```

// Declares object as a button
Button getAnswer = (Button)findViewById(R.id.getAnswerButton);

// Set listener for button.
getAnswer.setOnClickListener(getAnswerListener);
}

// Define OnClickListener for getAnswer button.
private OnClickListener getAnswerListener = new OnClickListener() {
    public void onClick(View v){ ... }
}

```

declares object as a button

name of button object

casting as a button

find view of button in layout

id of button

Set listener on button object

Listener object

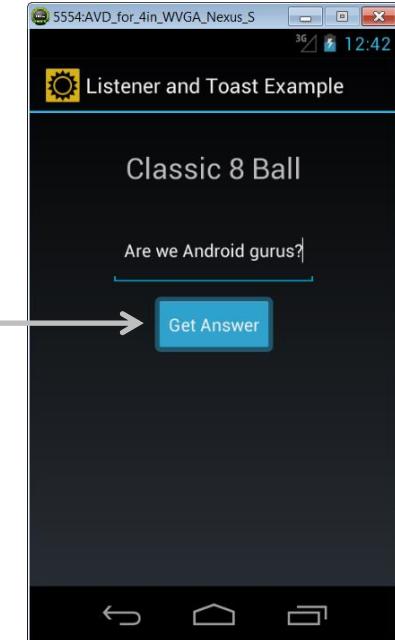
onClick method

call back code

View object parameter

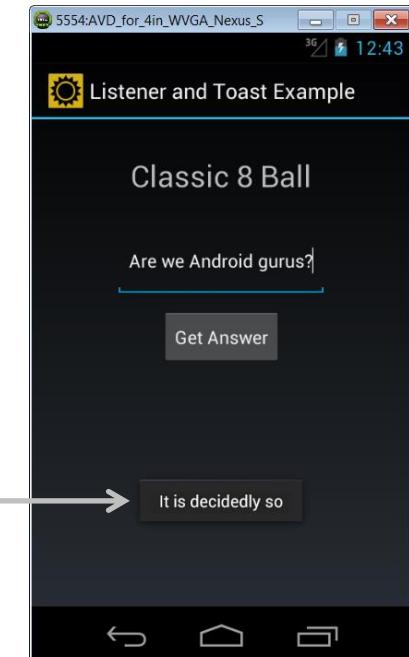
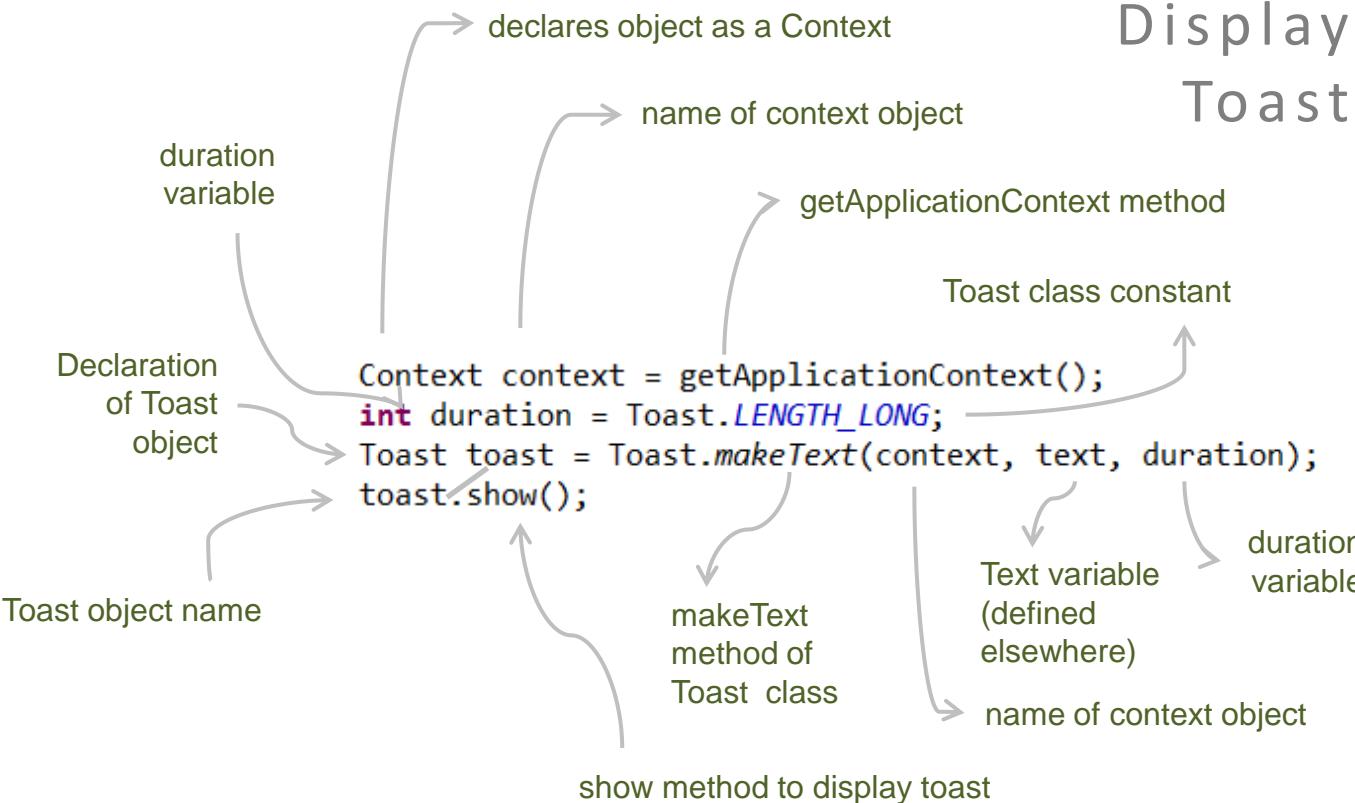
OnClickListener object

new instance of object



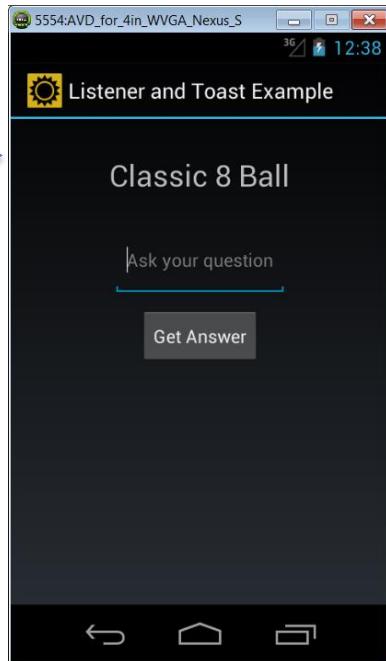
Standard sequence of statements used for setting up listeners for user interaction with screen objects.

# Display Toast

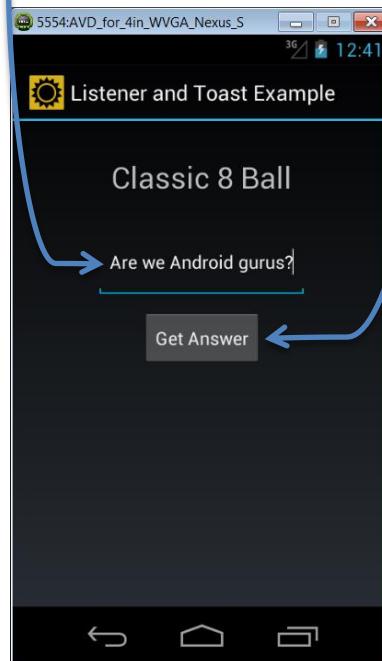


# Listeners and Toasts

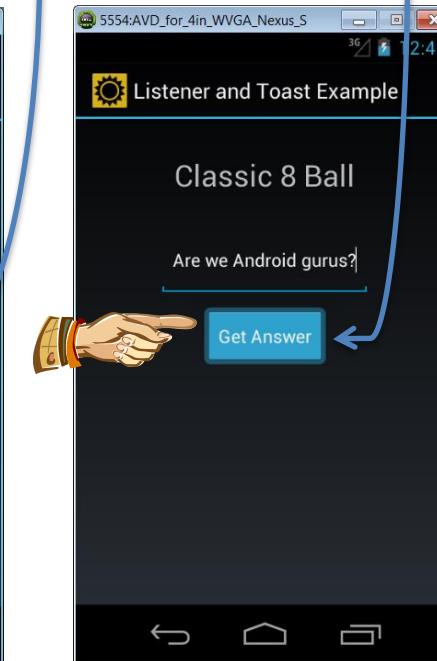
Set up display



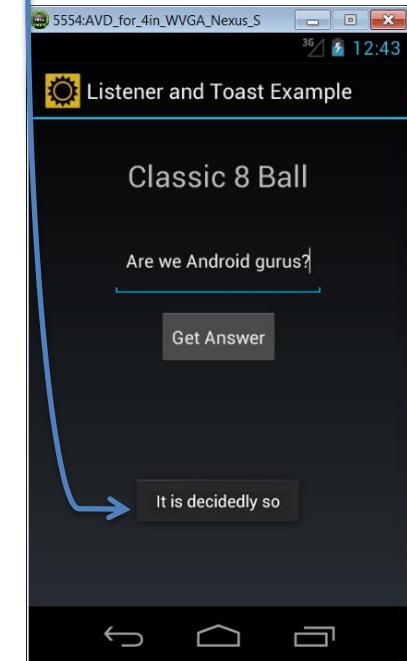
Accept input



Set up listener for button click

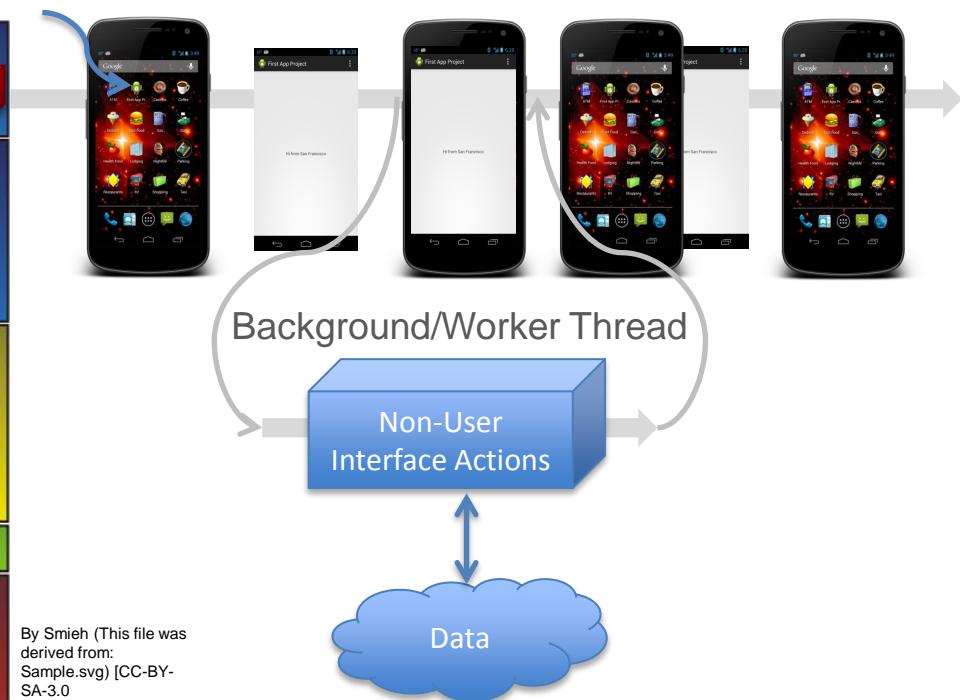
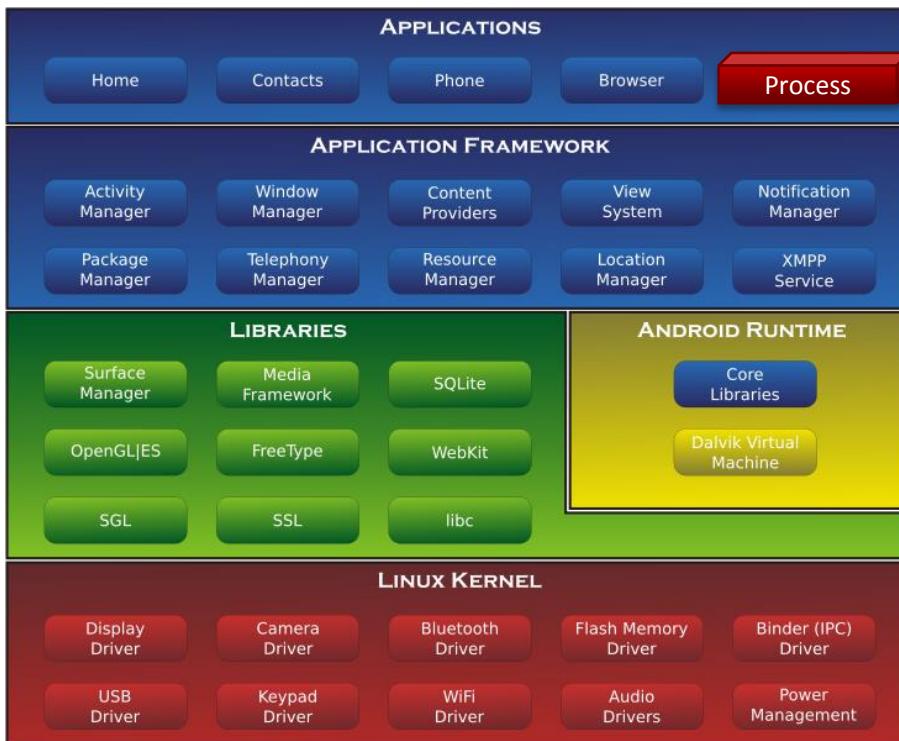


Respond to click listener

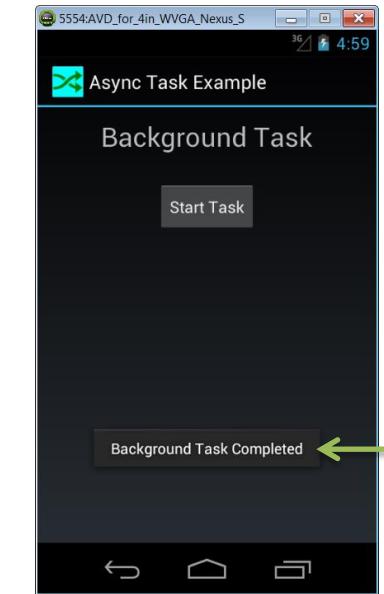
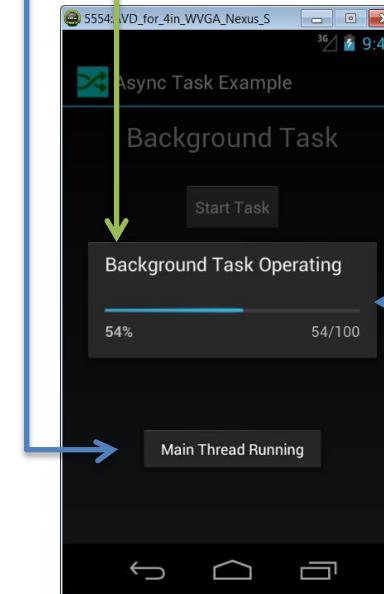
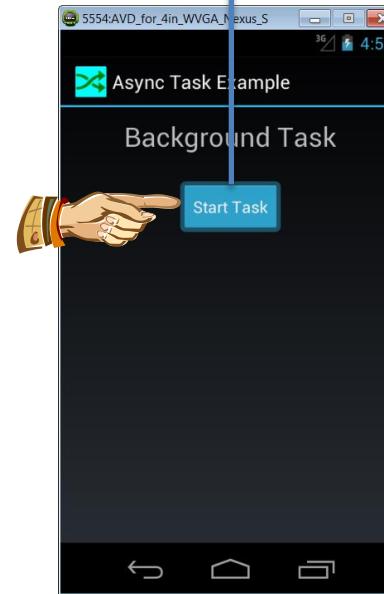
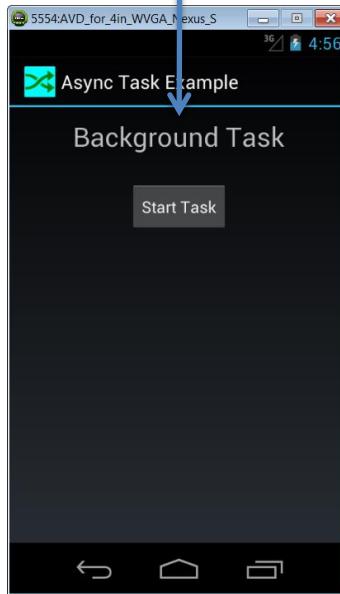


Toast answer

# App Threads



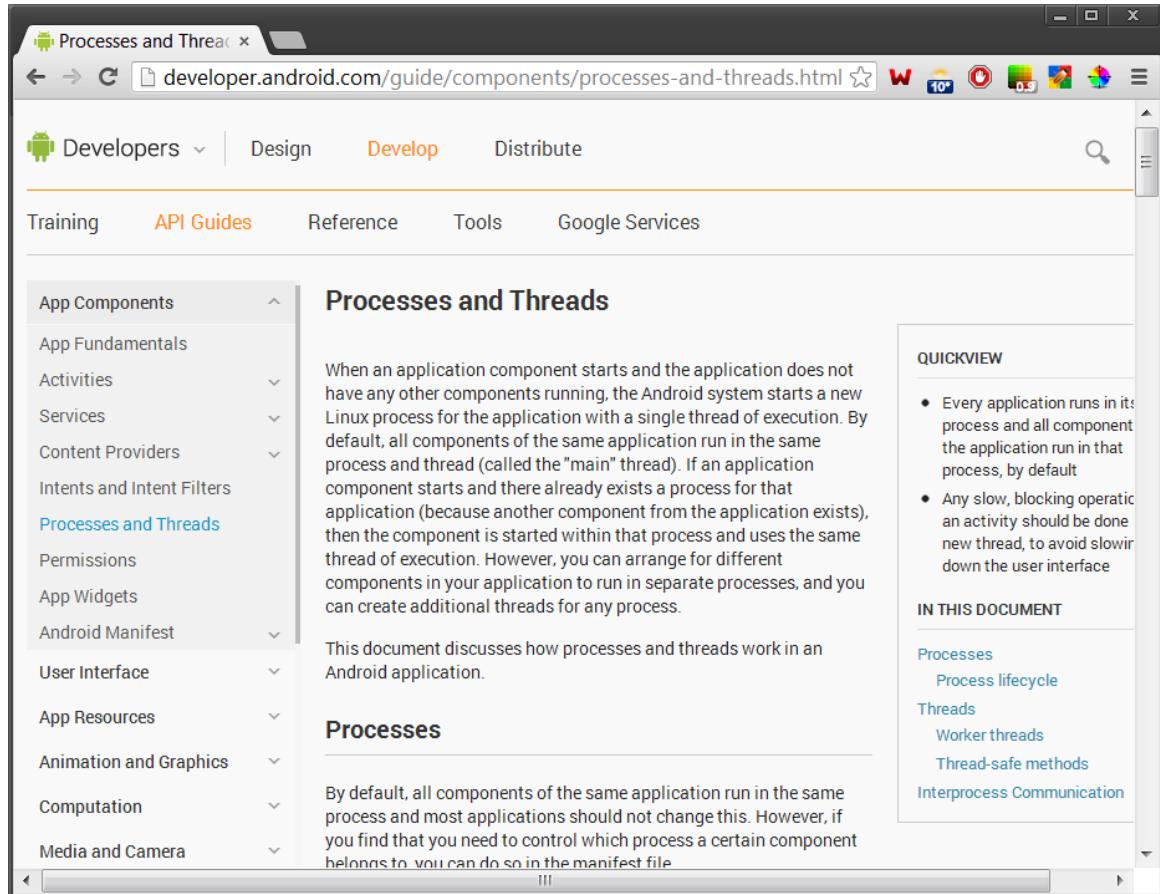
By Smieh (This file was derived from: Sample.svg) [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

AsyncTask  
Background  
ThreadMain  
ThreadBackground Task  
Example

# Processes and Threads

<http://developer.android.com/guide/components/processes-and-threads.html>

Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).



The screenshot shows a web browser window displaying the 'Processes and Threads' page from the official Android developer documentation at [developer.android.com/guide/components/processes-and-threads.html](http://developer.android.com/guide/components/processes-and-threads.html). The browser interface includes a header with the Android logo and 'Developers' tab, a navigation bar with 'Design', 'Develop', and 'Distribute' tabs, and a main content area with a sidebar and a main article.

**Side Navigation Bar:**

- App Components
- App Fundamentals
- Activities
- Services
- Content Providers
- Intents and Intent Filters
- Processes and Threads** (selected)
- Permissions
- App Widgets
- Android Manifest
- User Interface
- App Resources
- Animation and Graphics
- Computation
- Media and Camera

**Main Content Area:**

## Processes and Threads

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same application run in the same process and thread (called the "main" thread). If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution. However, you can arrange for different components in your application to run in separate processes, and you can create additional threads for any process.

This document discusses how processes and threads work in an Android application.

### Processes

By default, all components of the same application run in the same process and most applications should not change this. However, if you find that you need to control which process a certain component belongs to, you can do so in the manifest file.

**QUICKVIEW**

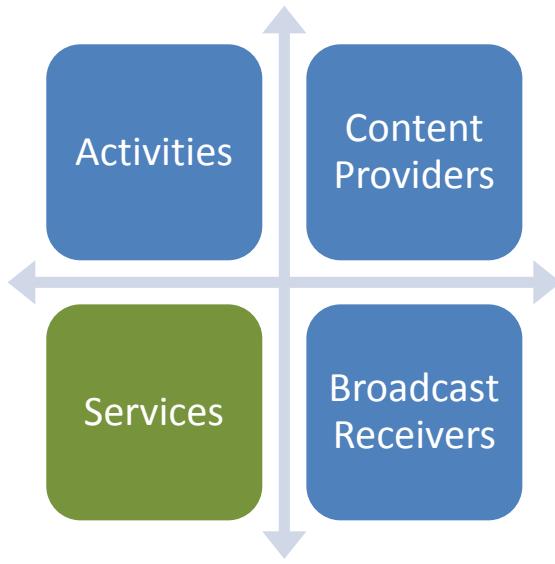
- Every application runs in its process and all components of the application run in that process, by default
- Any slow, blocking operation in an activity should be done in a new thread, to avoid slowing down the user interface

**IN THIS DOCUMENT**

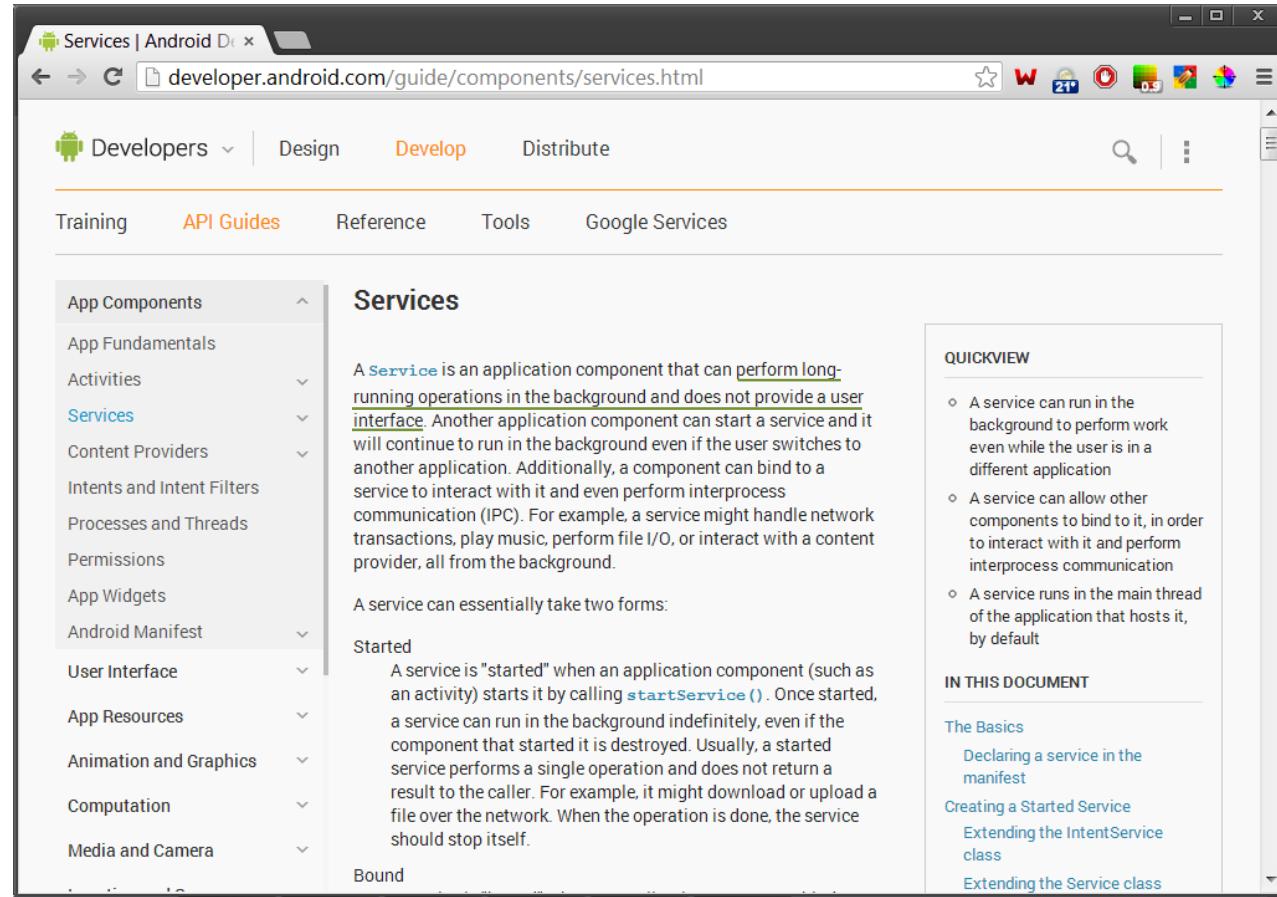
- Processes
- Process lifecycle
- Threads
- Worker threads
- Thread-safe methods
- Interprocess Communication

# Services

<http://developer.android.com/guide/components/services.html>



Portions of this page are reproduced from work created and shared by the Android Open Source Project and used according to terms described in the [Creative Commons 2.5 Attribution License](#).



The screenshot shows a web browser window displaying the Android Developers website at developer.android.com/guide/components/services.html. The page has a dark header with the Android logo and the word 'Develop'. Below the header, there's a navigation bar with tabs for 'Training', 'API Guides' (which is active), 'Reference', 'Tools', and 'Google Services'. On the left, there's a sidebar titled 'App Components' with a tree view containing 'App Fundamentals', 'Activities', 'Services' (which is expanded to show 'Content Providers', 'Intents and Intent Filters', 'Processes and Threads', 'Permissions', 'App Widgets', and 'Android Manifest'), 'User Interface', 'App Resources', 'Animation and Graphics', 'Computation', and 'Media and Camera'. The main content area is titled 'Services' and contains the following text:

A **Service** is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

A service can essentially take two forms:

**Started**

A service is "started" when an application component (such as an activity) starts it by calling `startService()`. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

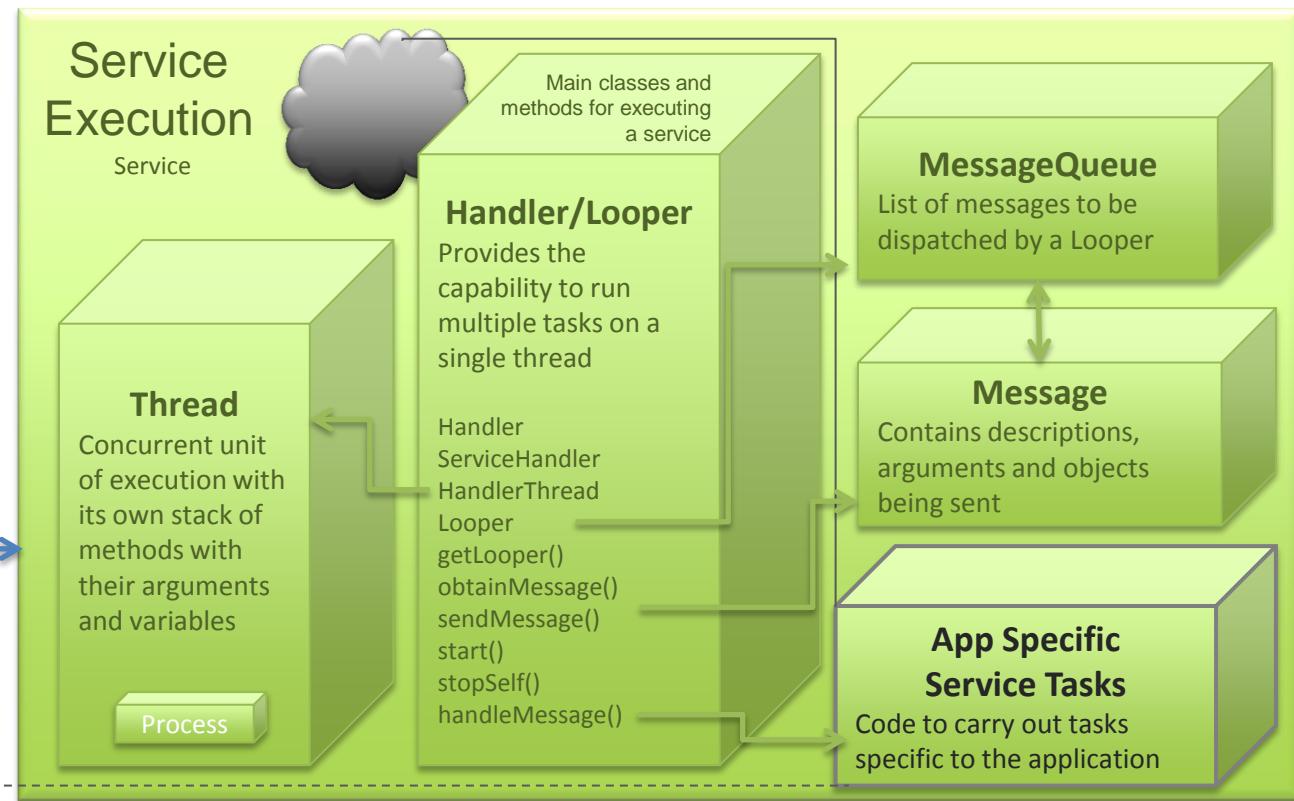
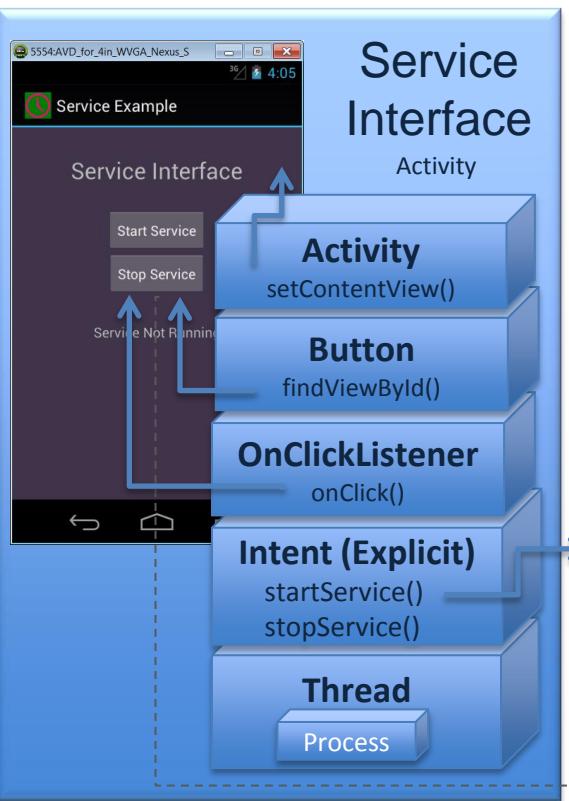
**Bound**

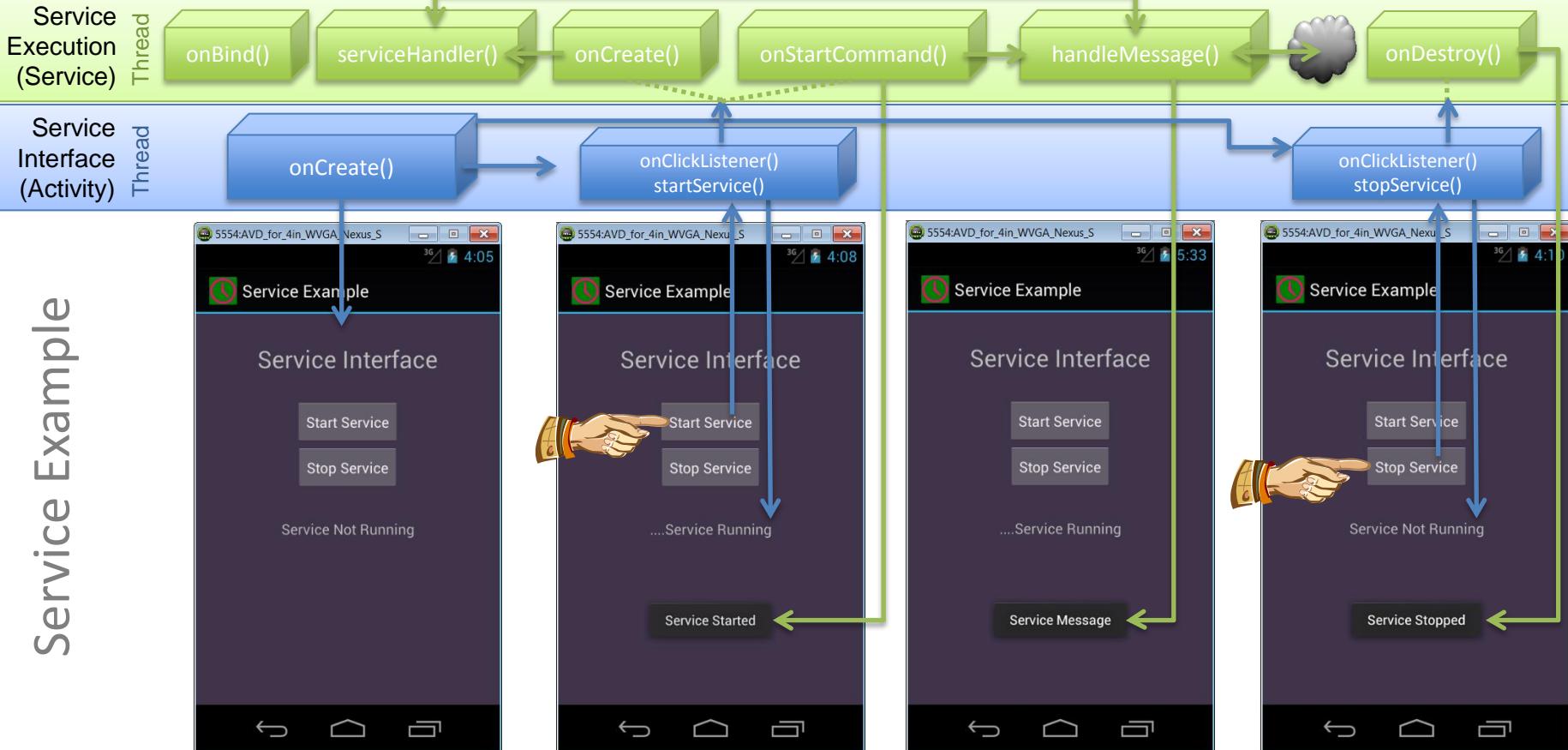
In the bottom right corner of the main content area, there's a 'QUICKVIEW' section with three bullet points:

- A service can run in the background to perform work even while the user is in a different application
- A service can allow other components to bind to it, in order to interact with it and perform interprocess communication
- A service runs in the main thread of the application that hosts it, by default

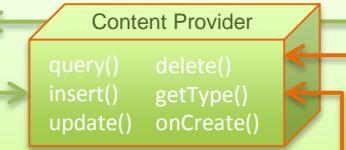
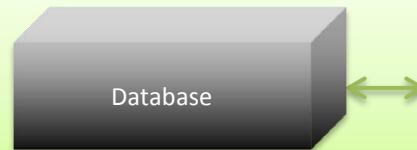
Below the main content area, there's a 'IN THIS DOCUMENT' section with links to 'The Basics', 'Declaring a service in the manifest', 'Creating a Started Service', 'Extending the IntentService class', and 'Extending the Service class'.

# Objects Used to Implement a Service

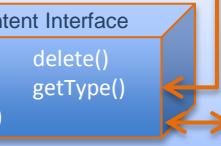




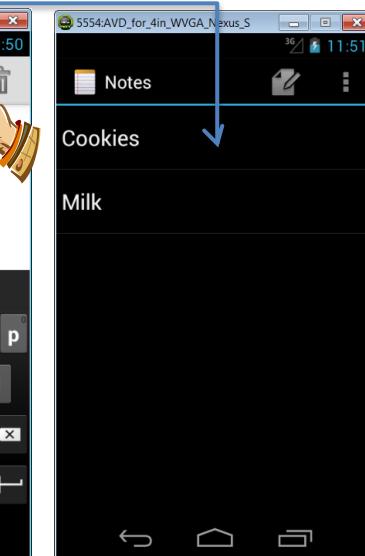
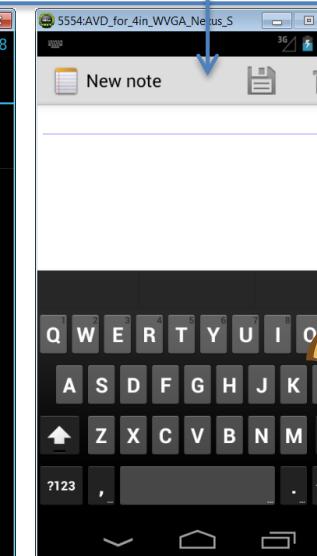
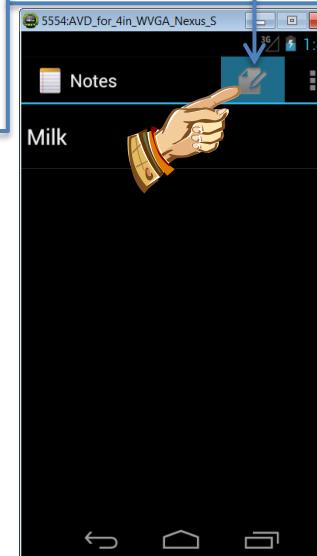
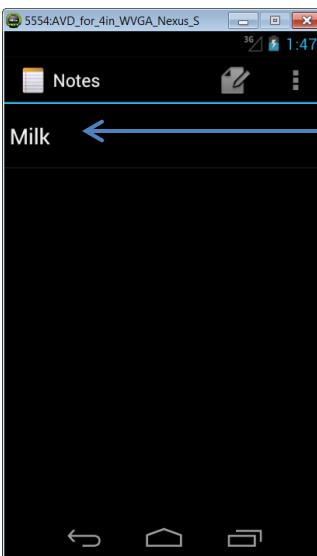
Content Manager  
(Thread)



Content User Thread

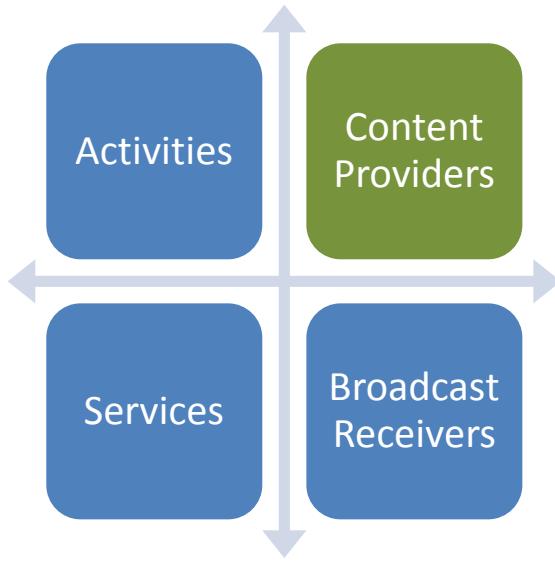


NotePad App  
Content Provider



# Content Providers

<http://developer.android.com/guide/topics/providers/content-providers.html>



## Use when your app needs to:

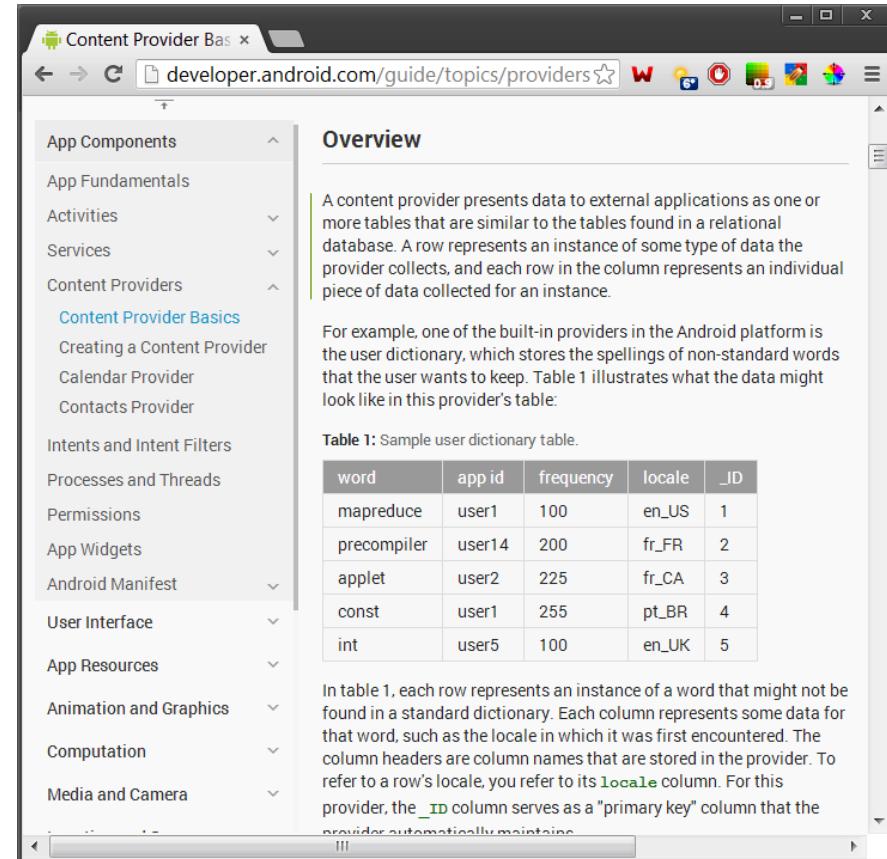
- Share data with other apps
- Provide custom search suggestions
- Copy and paste complex data or files from your app to other apps

## Custom Search

<http://developer.android.com/guide/topics/search/adding-custom-suggestions.html>

## Copy and Paste

<http://developer.android.com/guide/topics/text/copy-paste.html>



**Overview**

A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database. A row represents an instance of some type of data the provider collects, and each row in the column represents an individual piece of data collected for an instance.

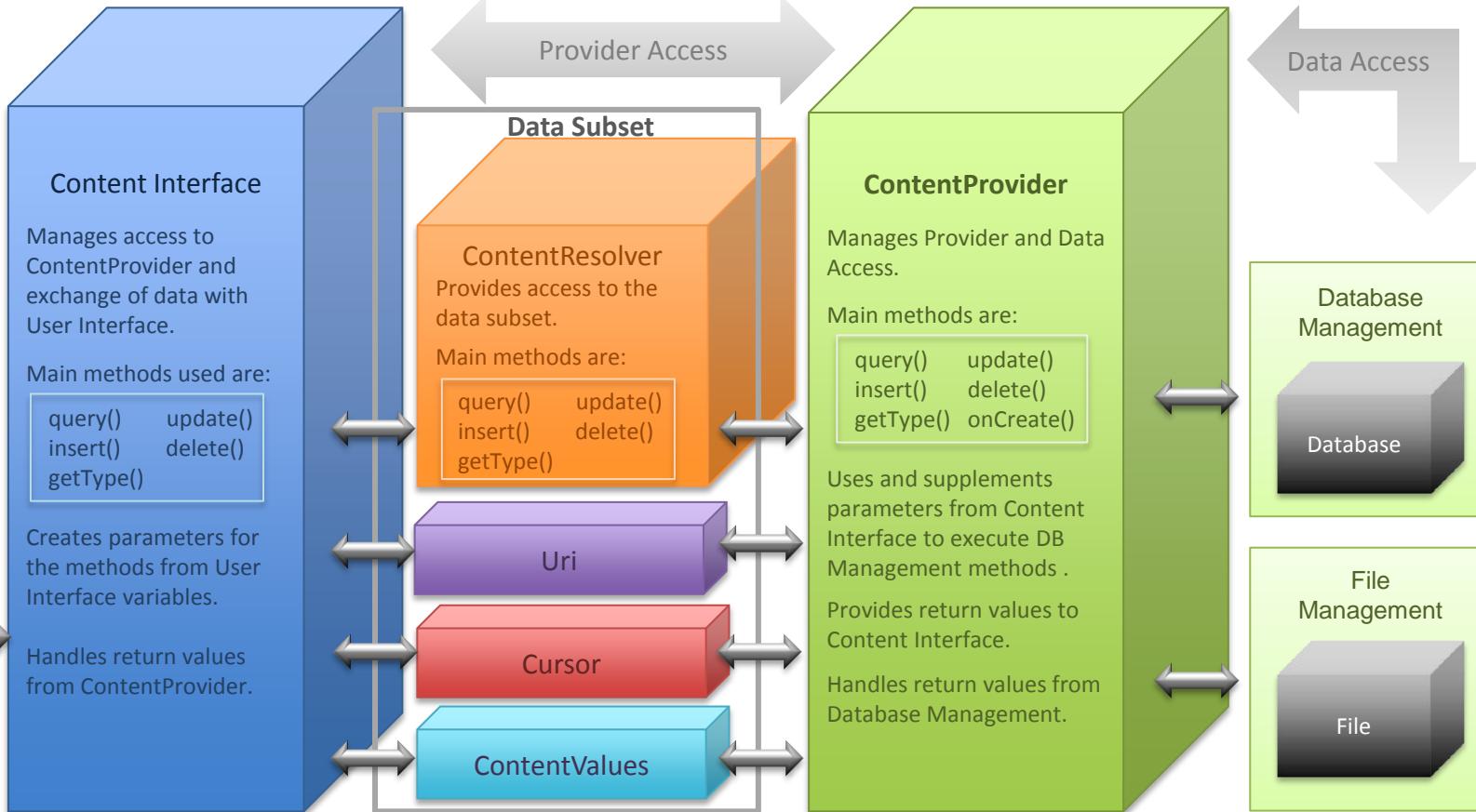
For example, one of the built-in providers in the Android platform is the user dictionary, which stores the spellings of non-standard words that the user wants to keep. Table 1 illustrates what the data might look like in this provider's table:

**Table 1: Sample user dictionary table.**

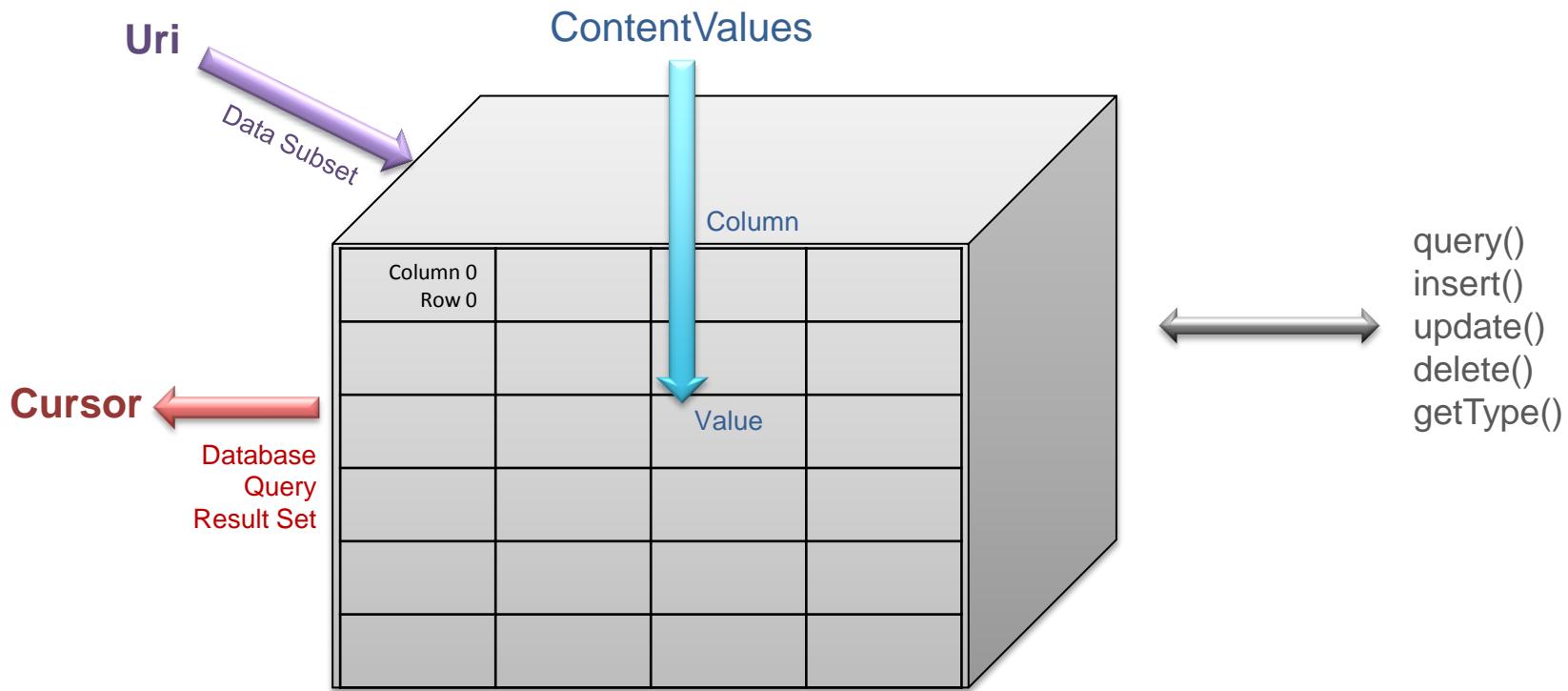
word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

In table 1, each row represents an instance of a word that might not be found in a standard dictionary. Each column represents some data for that word, such as the locale in which it was first encountered. The column headers are column names that are stored in the provider. To refer to a row's locale, you refer to its `locale` column. For this provider, the `_ID` column serves as a "primary key" column that the provider automatically maintains.

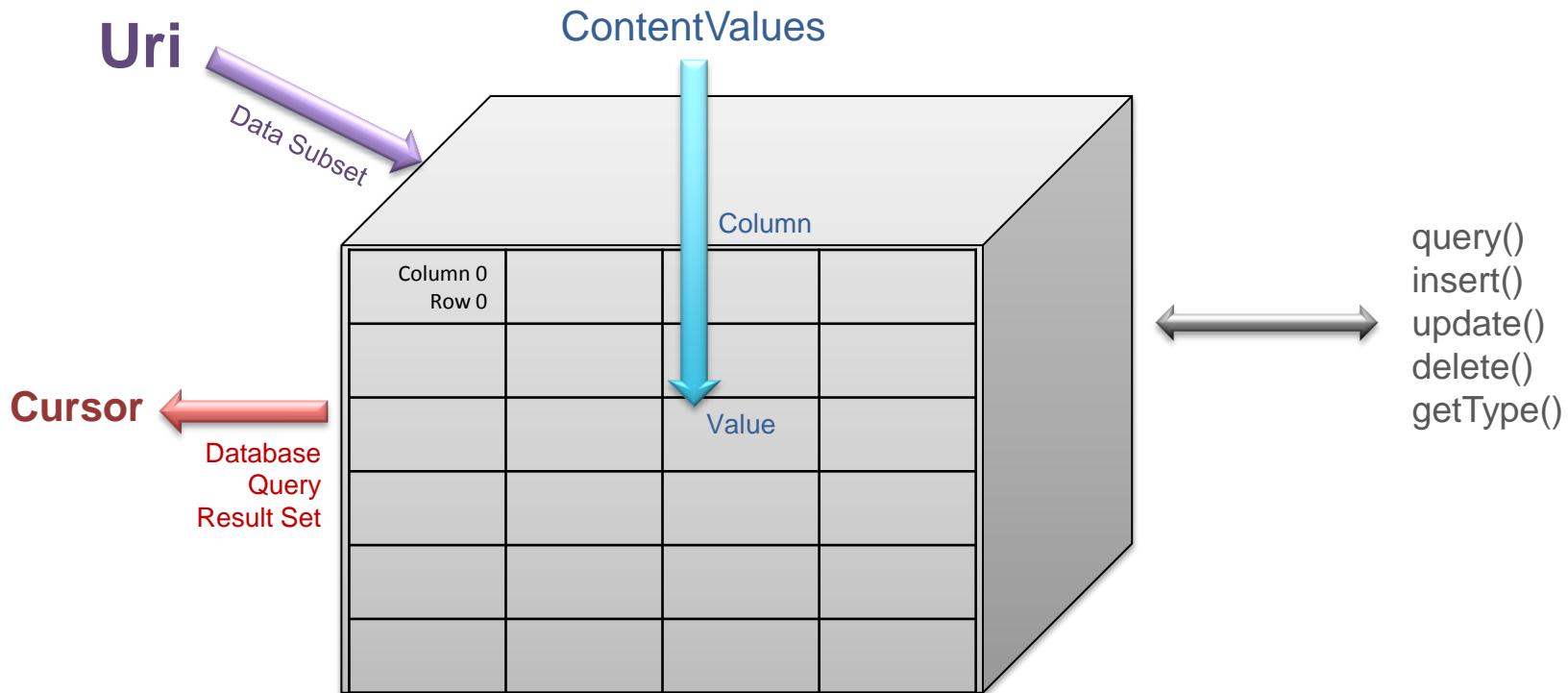
# Content Provider Classes and Objects



# Data Specification



# URI (Uniform Resource Identifier)



# URI Format

**scheme:scheme-specific-part**

Examples:

Web address	<code>http://example.org/absolute/URI/with/absolute/path/to/resource.txt</code>
File transfer	<code>ftp://example.org/resource.txt</code>
Resource name	<code>urn:isbn:0-486-27557-4</code>
Android content provider	<code>content://com.google.provider.NotePad/notes/4</code>

[http://en.wikipedia.org/wiki/Uniform\\_resource\\_identifier](http://en.wikipedia.org/wiki/Uniform_resource_identifier)

[http://en.wikipedia.org/wiki/URI\\_scheme](http://en.wikipedia.org/wiki/URI_scheme)

# Content Provider URI

content://authority/path/id

Identifies a data subset

Used for methods:  
query()  
insert()  
update()  
delete()  
getType()

```
// Define URI component strings.  
String SCHEME      = ContentResolver.SCHEME.CONTENT;    // Scheme (content://)  
String AUTHORITY   = "com.google.provider.NotePad";    // Authority for controlling access  
String PATH_NOTE_ID = "/notes";                         // Path to identify an individual group  
String SAMPLE_Id   = "/4";                             // ID to specify a single row number  
  
// Defines a URI in the required format.  
// In this example the resulting string is "content://com.google.provider.NotePad/notes/4"  
Uri SAMPLE_URI = Uri.parse(                                // Use parse() method to format a Uri  
    SCHEME +                                         // Scheme  
    AUTHORITY +                                       // Authority  
    PATH_NOTE_ID +                                     // Path - none, one or multiple paths  
    SAMPLE_ID);                                      // ID - optional
```

## Description

```
UriMatcher(int code)
- code to match for the root URI

addURI(String authority, String path, int code)
- authority controls access to data
- path identifies individual elements
  * used to match any character
  # used to match number
- code is the number returned on a match

match(Uri uri)      returns int code
- uri controls access to data
- code is the number returned on a match
```

## Steps

### Create

```
// Create a new instance of UriMatcher object
// NO_MATCH = -1 for no matched nodes
private static final UriMatcher sUriMatcher;
sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

### Setup

```
// Add entries to UriMatcher:
//           authority          path        code
// -----
sUriMatcher.addURI(NotePad.AUTHORITY, "notes", NOTES );
sUriMatcher.addURI(NotePad.AUTHORITY, "notes/#", NOTE_ID);
```

### Switch

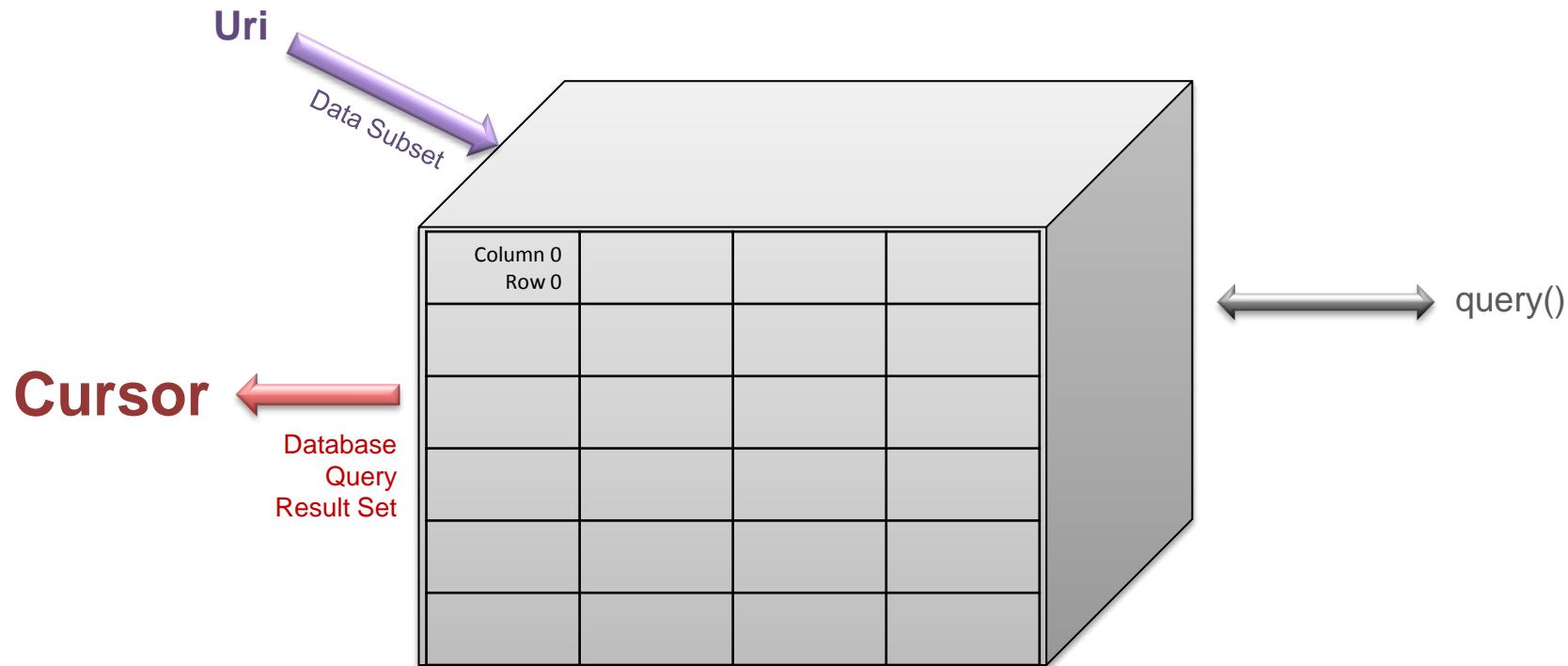
```
// Use match() method to retrieve code for the uri
switch (sUriMatcher.match(uri)) {

    // Take action based on code
    case NOTES:
        qb.setProjectionMap(sNotesProjectionMap);
        break;
    case NOTE_ID:
        qb.setProjectionMap(sNotesProjectionMap);
    .
    .
}
```

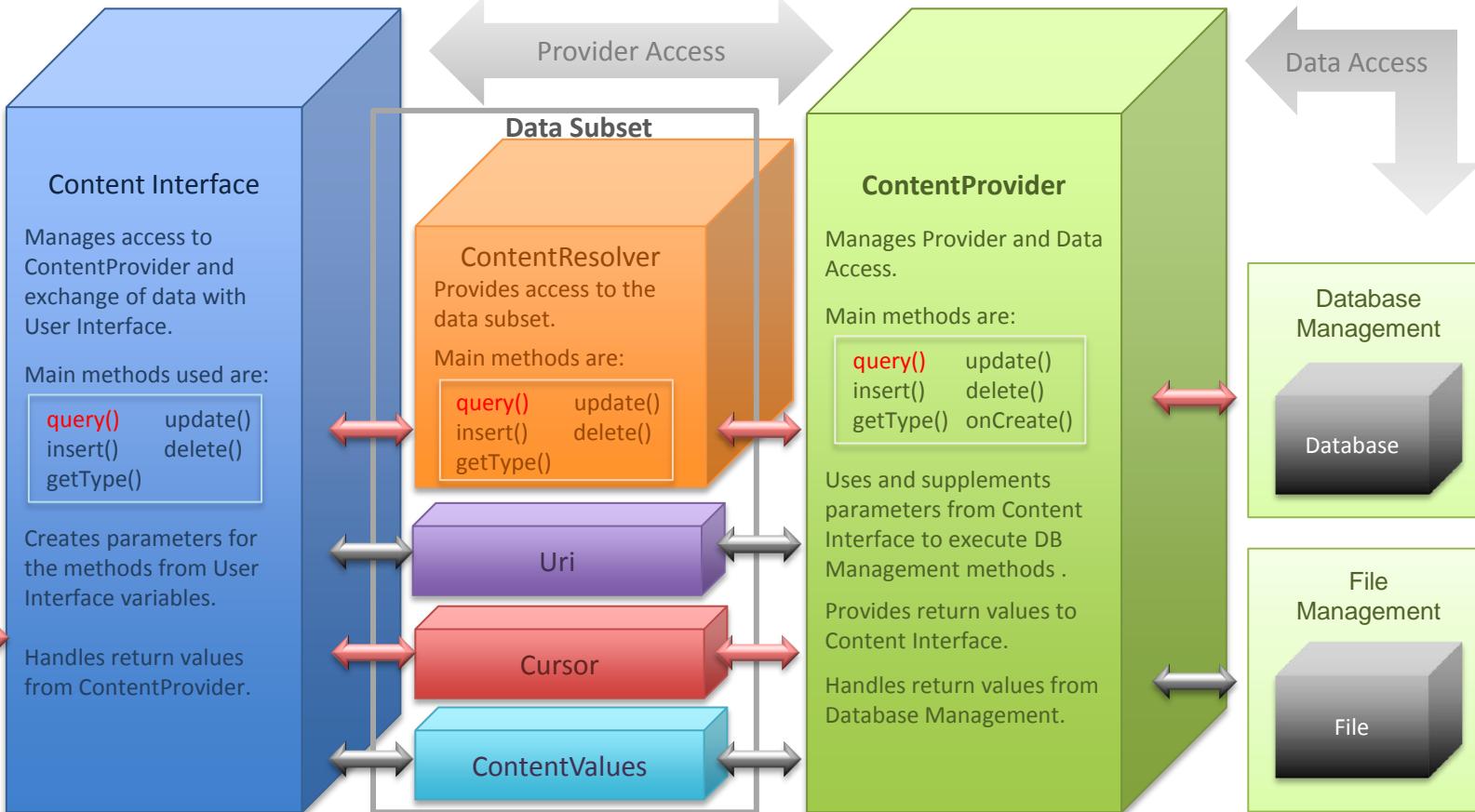
## Example

# URI Matcher

# Cursor Use in Content Providers



# Content Provider Classes and Objects



# Cursor Class and Objects

Provide read/write access to a result set returned from a database query.

move(int offset)  
moveToFirst()  
moveToLast()  
moveToNext()  
moveToPrevious()  
moveToPosition(int position)

Keeps track of current row number

Column 0 Row 0			

getDouble(int columnIndex)  
getFloat(int columnIndex)  
getLong(int columnIndex)  
getShort(int columnIndex)  
getString(int columnIndex)  
getDouble(int columnIndex)

columnIndex

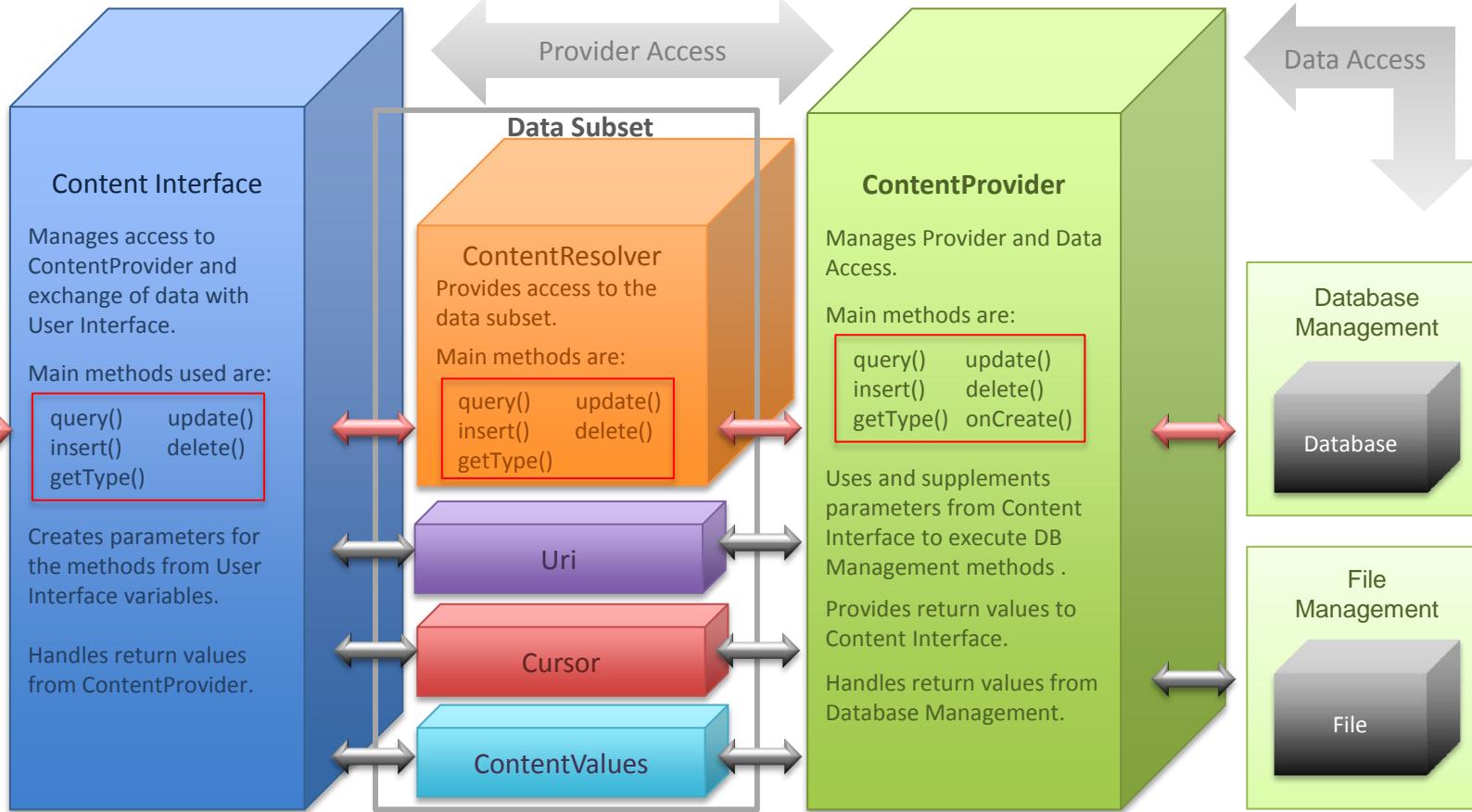
Used for methods:  
query()

Cursor c (*return from a query*)

Cursor is used by other classes and methods such as cursor adapters for list displays.

getColumnName(String columnName)  
getColumnCount()  
getColumnNames()  
getCount() (*returns total number of rows*)  
setNotificationUri(ContentResolver cr, Uri uri)

# Content Provider Methods



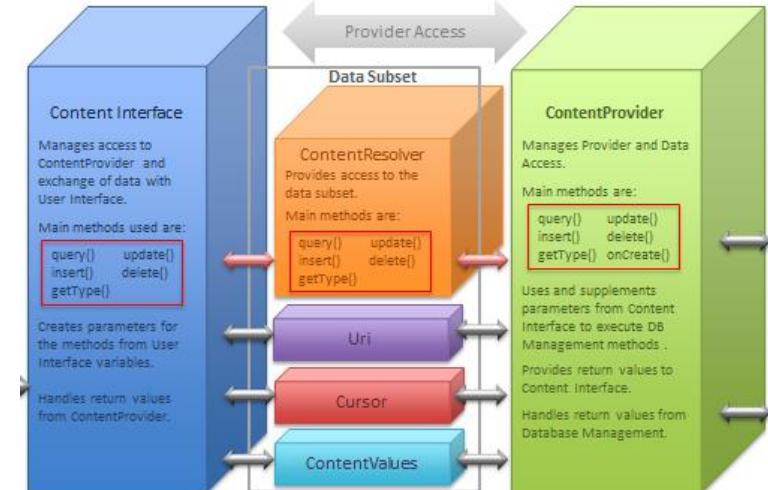
## onCreate() Method

Content Interface uses a ContentResolver object to access some data for the first time

ContentResolver tries to access ContentProvider for first time

ContentProvider object created

ContentProvider onCreate method invoked



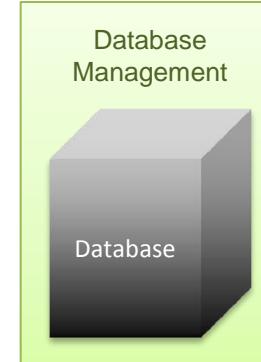
### ContentProvider onCreate() method

- Initialize variables
- Open files/database
- Keep minimal

# Key Methods

## Parameter Use

 Applicable

 Can be null


Parameters	query()	insert()	update()	delete()	getType()
Uri					
Projection					
Selection Clause					
Selection Arguments					
Sort Order					
ContentValues					
Cancellation Signal (Introduced in API 16)					
	Cursor	Uri	int number of rows affected	int number of rows affected	String of MIME type

## Return Use

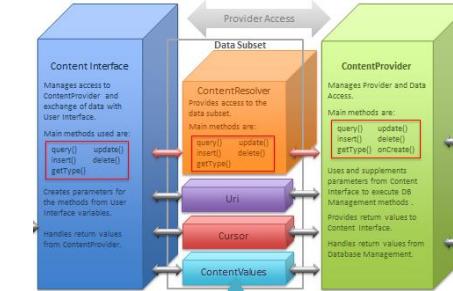
# ContentValues

Identifies column/value pairs

```
// Defines an object to contain new values to insert.
ContentValues mValues = new ContentValues();
```

```
// Sets values in the ContentValues object
// using the put() method of the ContentValues class
// with Column Name and Value string as parameters.
//
```

	Key (Column Name)	Value
-----		
mValues.put( NotePad.Notes.COLUMN_NAME_TITLE, "Milk" );		
mValues.put( NotePad.Notes.COLUMN_NAME_TYPE, "Soy" );		



Column 0	Row 0	title	type
		Milk	Soy

// Sets title Milk  
// Sets type Soy

Used for methods:  
insert()  
update()

# Selection Clause and Arguments

```
// Defines the criteria for which rows to return,
// formatted as an SQL WHERE clause (excluding the WHERE itself).
// Passing null will return all rows for the given table.
// Question marks will be replaced with corresponding entry in the
// selection arguments parameter below.
```

```
String mSelectionClause =
    "NotePad.Notes.COLUMN_NAME_TITLE=? AND
     NotePad.Notes.COLUMN_NAME_TYPE=?";
```

// Define a string  
// Title column selection  
// Type column selection

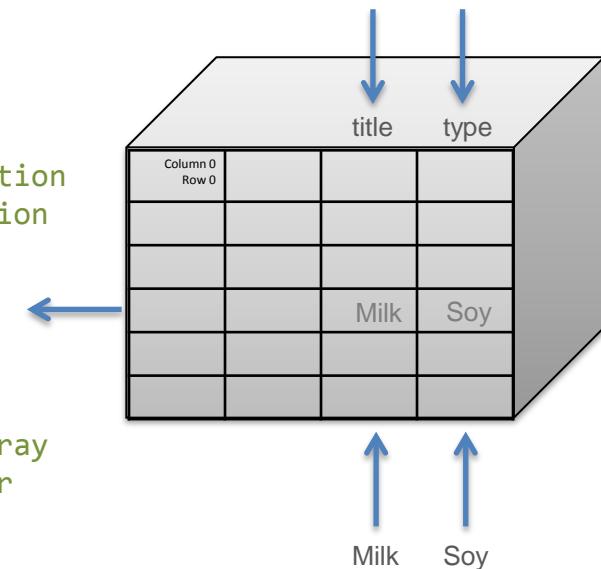
```
// Defines values input by user to be paired
// with selection clause fields above.
```

```
String[] mSelectionArguments = new String[] {
    "Milk",
    "Soy"};
```

// Create a string array  
// Title input by user  
// Type input by user

Identifies criteria for access operations

Used for methods:  
 query()  
 update()  
 delete()



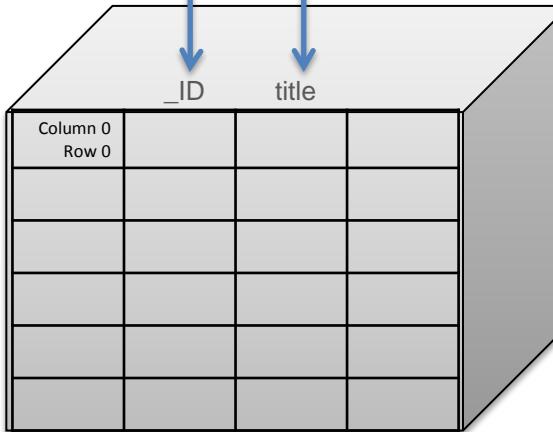
# Projection

Identifies columns for data retrieval

Used for methods:  
query()

// Defines the columns to be returned for each row returned from a query.

```
String[] mProjection = new String[] {  
    NotePad.Notes._ID,  
    NotePad.Notes.COLUMN_NAME_TITLE  };  
// Create a string array  
// Column = "_ID"  
// Column = "title"
```



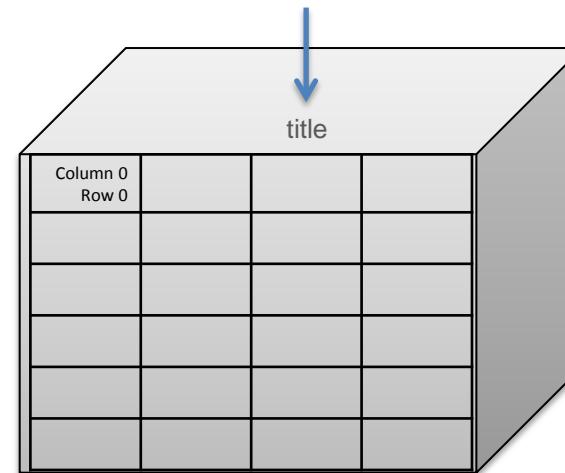
## Sort Order

Identifies column for sort

Used for methods:  
query()

// Specifies the column to be used to sort query results.

```
String mSortOrder = "NotePad.Notes.COLUMN_NAME_TITLE"           // Title column sort
```



# Cancellation Signal

Cancels operation in progress

Used for methods:  
query()

```
// Create a CancellationSignal object.  
// Introduced in API 16.  
// If this parameter is used, you must also implement  
// a query() method without it to support previous versions.  
CancellationSignal mCancellationSignal = new CancellationSignal() // Instantiate CancellationSignal  
.  
.  
.  
  
// Cancel the query operation.  
mCancellationSignal.cancel();
```

# MIME Type

Provides information about the file type

Used for methods:  
getType()

String in the format: **type/subtype(optional-parameters)**

## Standard

Types: application, audio, image, message, text, video, . . .

Text subtype examples: calendar, plain, html, xml

Example: “text/html”

[http://en.wikipedia.org/wiki/Internet\\_media\\_type](http://en.wikipedia.org/wiki/Internet_media_type)

<http://www.iana.org/assignments/media-types>

## Android Vendor-Specific Custom

Type for multiple rows: vnd.android.cursor.dir

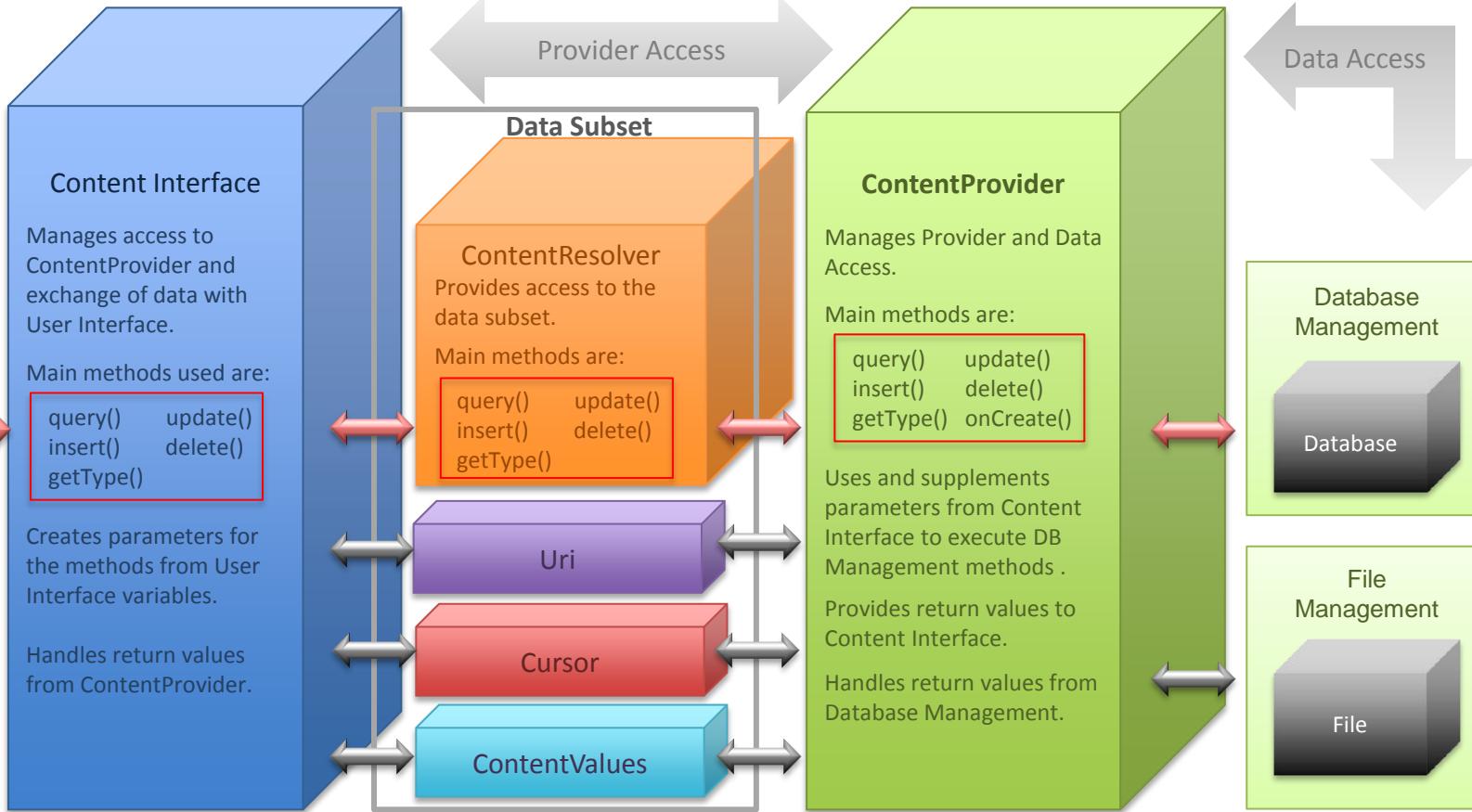
Type for Single row: vnd.android.cursor.item

Subtype example: vnd.google.note

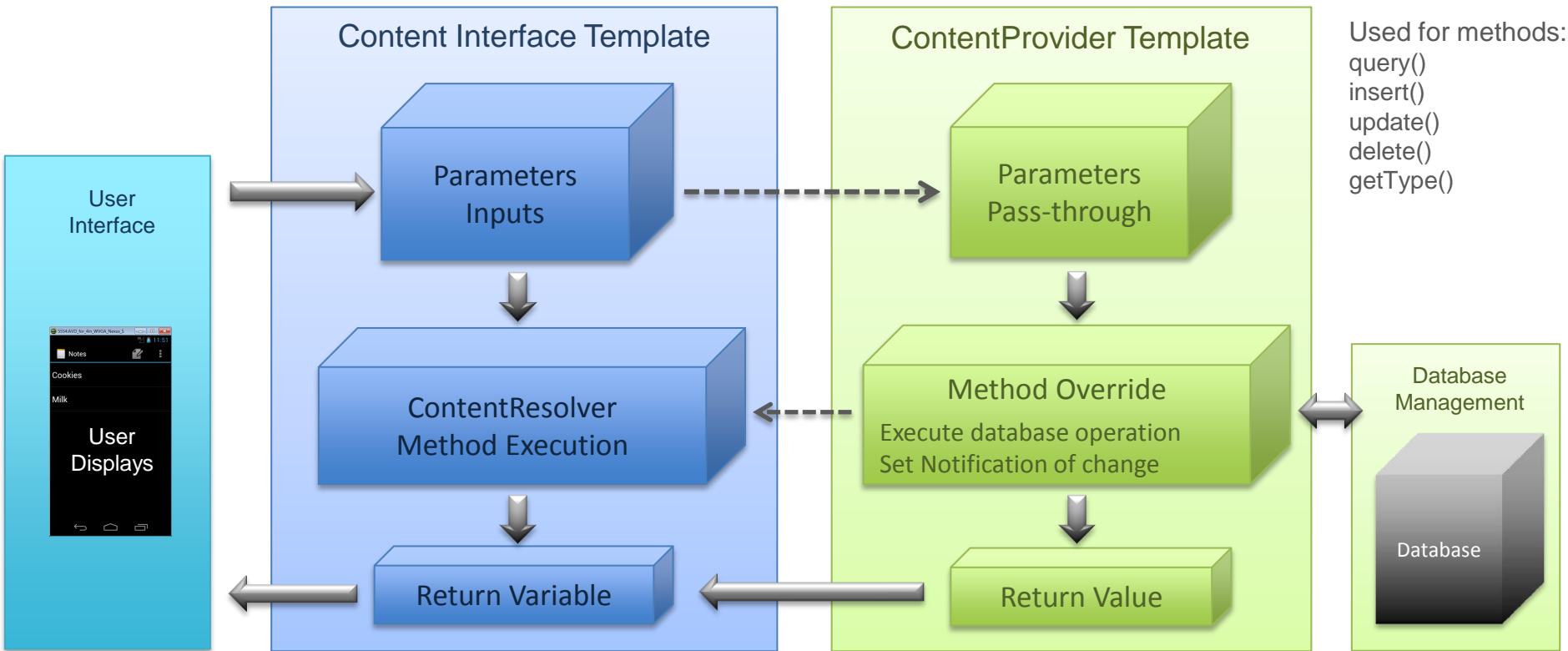
Example: “vnd.android.cursor.dir/vnd.google.note”

<http://developer.android.com/guide/topics/providers/content-provider-basics.html#MIMETypeReference>

# Content Provider Methods



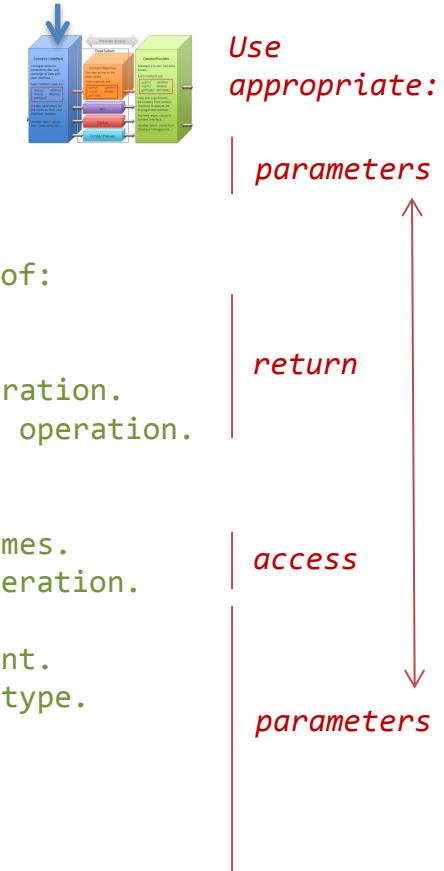
# Content Interface and Provider Method Templates



# Content Interface Method Template

```
// Define parameters needed for access. (see parameter specific graphics)
...
// Define a variable to hold return depending on the type of access. Use one of:
Cursor mReturn = new Cursor();           // Define Cursor for query operation.
Uri    mReturn = new Uri();                // Define Uri for insert operation.
int    mReturn = 0;                      // Define int for update or delete operation.
String mReturn = "";                     // Define MIME type String for getType operation.

// Execute operation and assign return result.
// Replace access with one of query, insert, update, delete, getType method names.
mReturn = getContentResolver().access( // Get Content Resolver and perform operation.
    mUri,                                // Parameter values - which are set by
    mProjection,                         // code executed prior to this statement.
    mSelectionClause,                   // Use parameters depending on access type.
    mSelectionArguments,                // Some parameters can be null.
    mSortOrder,
    mValues,
    mCancellationSignal);
```



# Key Methods

## Parameter Use

 Applicable

 Can be null


Parameters	query()	insert()	update()	delete()	getType()
Uri					
Projection					
Selection Clause					
Selection Arguments					
Sort Order					
ContentValues					
Cancellation Signal (Introduced in API 16)					
	Cursor	Uri	int number of rows affected	int number of rows affected	String of MIME type

## Return Use

# Key Methods

## Parameter Use

 Applicable

 Can be null


Parameters	query()	insert()	update()	delete()	getType()
Uri					
Projection					
Selection Clause					
Selection Arguments					
Sort Order					
ContentValues					
Cancellation Signal (Introduced in API 16)					
	Cursor	Uri	int number of rows affected	int number of rows affected	String of MIME type

## Return Use

# ContentProvider Method Template

```
// Use UriMatcher to take actions based on the Uri. (see Uri Matcher graphic)
...
// Execute operation and return result.
// Replace return with one of Cursor, Uri, int, String return types.
// Replace access with one of query, insert, update, delete, getType method names.

@Override
public return access(Uri uri, String[] projection, String selectionClause, String[] selectionArgs, String sortOrder, ContentValues values)
{
    // Body of access method (see next graphic)
}
```



Use appropriate:

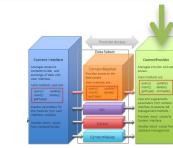
*Uri matching*

*return & access*

*parameters*

# ContentProvider Method Template

```
// Body of access method - code executed when the method is invoked.  
{  
    // Define a variable to hold result depending on the type of access. Use one of:  
    Cursor mReturn;          // Define Cursor for a query.  
    Uri    mReturn;          // Define Uri for an insert.  
    int    mReturn;          // Define integer for row count for an update or delete.  
    String mReturn;          // Define String for MIME type for a getType.  
  
    // Perform database access operation based on the method parameters  
    // which were given values by the Content Interface.  
    . . .                      // Prepare for access  
    mReturn = database.access(parameters . . .) // Access SQLite database & get return  
  
    // Perform Notification to set callbacks for access type.  
    mReturn.setNotificationUri(getContext().getContentResolver(), uri); // query  
    getContext().getContentResolver().notifyChange(uri, null); // update, delete, insert  
  
    // Return result  
    return mReturn           // Return result for access.  
}
```



Use appropriate:

return

access

notification

# Activity Lifecycle Methods and Typical Uses

## onCreate()

- Must be implemented
  - Call `setContentView()`
  - Initialize display layout
  - Initialize variables
  - Start background threads

## onResume()

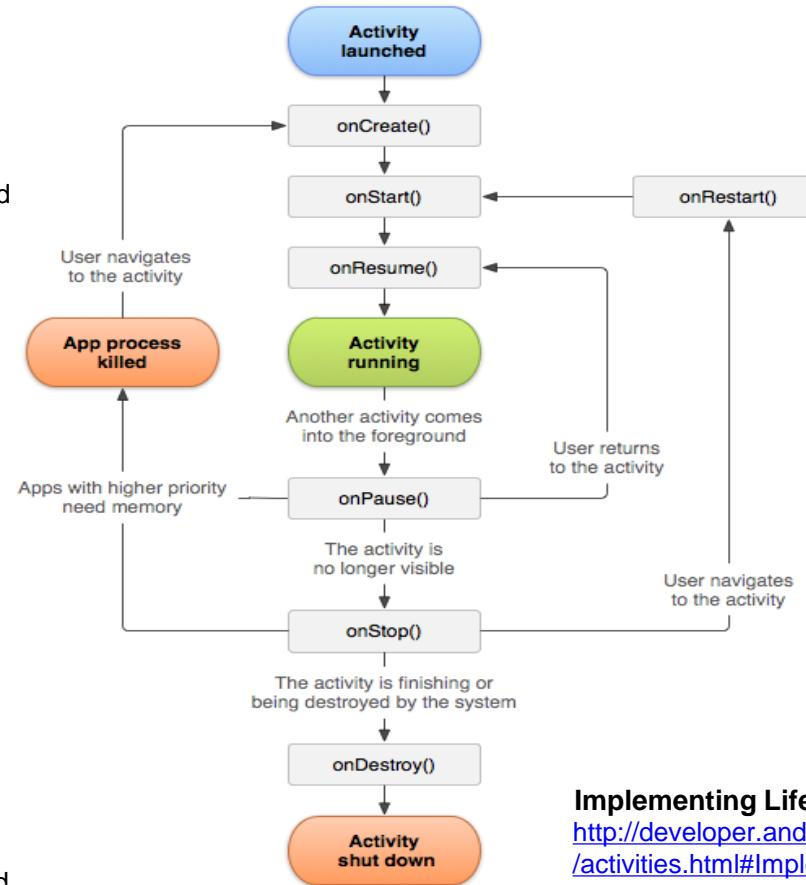
- Register Broadcast Receivers
  - Restore variables
  - Restart animations
  - Lightweight code

**onPause()**

- Commit changes
  - Save variables
  - Save preferences
  - Stop animations
  - Unregister Broadcast Receivers
  - Lightweight code

## **onDestroy()**

- Close databases
  - Release remaining resources
  - Stop background threads



## onStart()

- Register Broadcast Receiver
  - Reinitialize states

## onRestart()

- Open server connections
  - Allocate resources

## onStop()

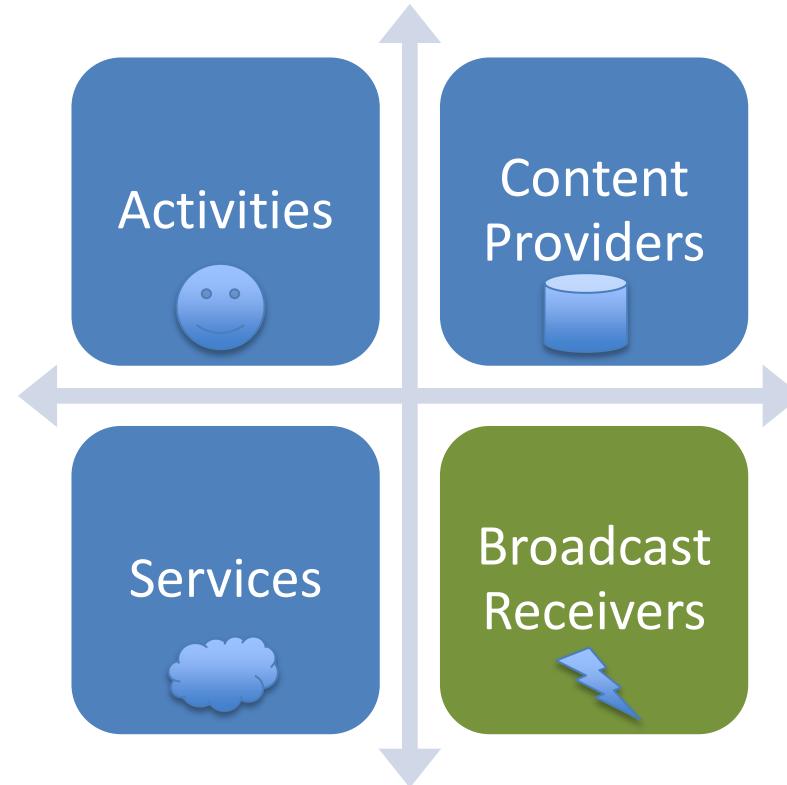
- Close server connections
  - Free resources
  - Unregister Broadcast Receiver

## Implementing Lifecycle Callbacks

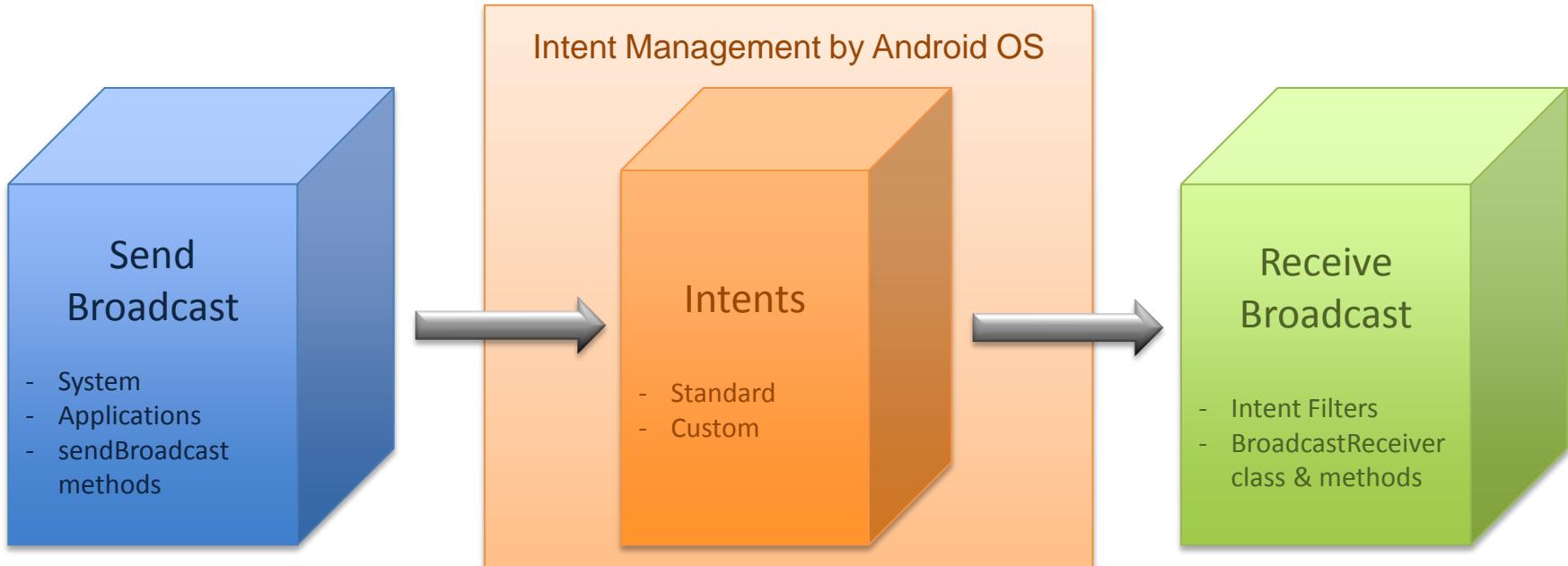
<http://developer.android.com/guide/components/activities.html#ImplementingLifecycleCallbacks>

Portions of this page are reproduced from work created and [shared by the Android Open Source Project](#) and used according to terms described in the [Creative Commons 2.5 Attribution License](#).

Broadcast  
Receivers



# Sending and Receiving Broadcasts



[http://developer.android.com/reference  
/android/content  
/Context.html#sendBroadcast\(android.content.Intent\)](http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent))

[http://developer.android.com  
/reference/android/content/Intent.html](http://developer.android.com/reference/android/content/Intent.html)

[http://developer.android.com/reference  
/android/content/BroadcastReceiver.html](http://developer.android.com/reference/android/content/BroadcastReceiver.html)

# Sending a Broadcast

## Variations on sendBroadcast()

**AsUser:** Allows you to specify the user the broadcast will be sent to.

**Ordered:** Intent is delivered to BroadcastReceivers one at a time.

**Sticky:** Intent stays around after the broadcast is complete

**receiverPermission:** Parameter that allows an optional required permission to be enforced.

## Simple sendBroadcast() example for a custom intent

```
Intent intent = new Intent(); // Instantiate Intent object
intent.setAction("com.donkcowan.specialaction"); // Set action using a custom intent
sendBroadcast(intent); // Send broadcast with intent
```

# Key Objects Used with Content Providers

