Mobile programming

Omid Jafarinezhad

Android 03

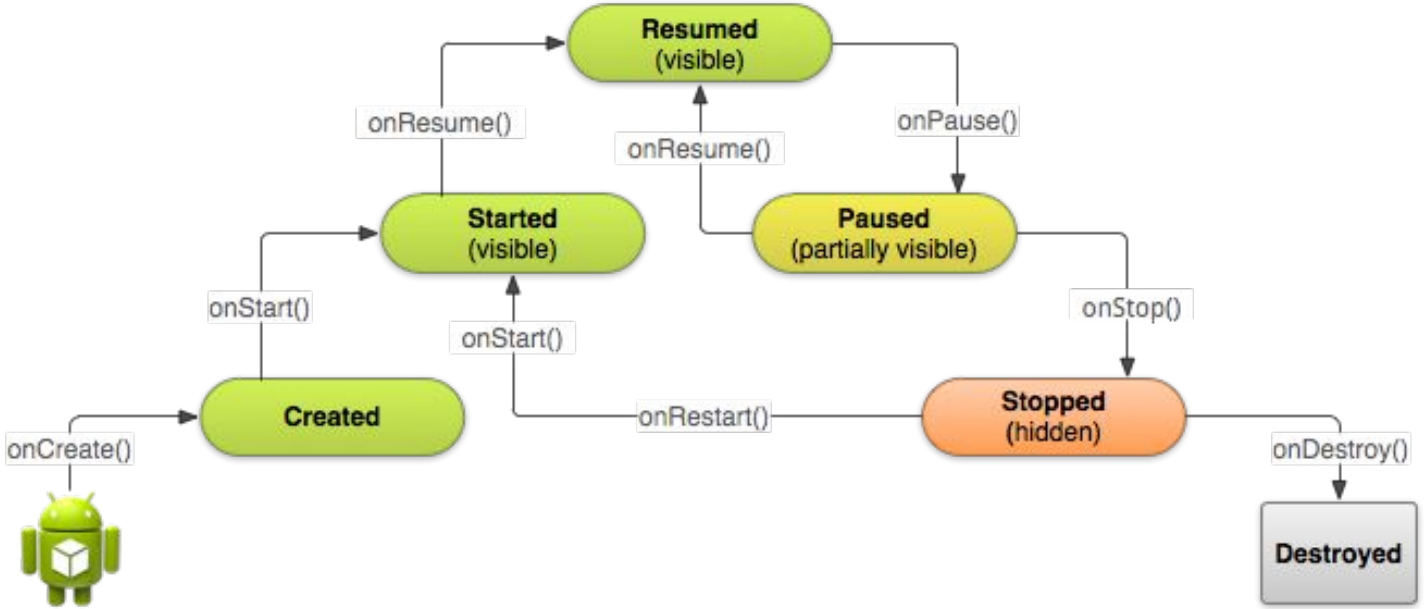
Managing the Activity Lifecycle

As a **user navigates through, out of, and back to your app**, the Activity instances in your **app transition between different states in their lifecycle**

If the user performs an action that **starts another activity or switches to another app**, the **system calls another set of lifecycle methods on your activity as it moves into the background** (where the activity is no longer visible, but the instance and its state remains intact)

Within the **lifecycle callback methods**, you can declare **how your activity behaves** when the user leaves and re-enters the activity

Understand the Lifecycle Callbacks



Understand the Lifecycle Callbacks

During the life of an activity, the system calls a core set of lifecycle methods in a sequence similar to a step **pyramid**

As the system creates **a new activity instance**, each callback method **moves the activity state one step toward the top**. The **top of the pyramid** is the point at which the **activity is running in the foreground** and the user can interact with it

As the user begins to **leave the activity**, the system calls other methods that move the **activity state back down the pyramid** in order to dismantle the activity

Why?

Implementing your activity lifecycle methods properly ensures your app behaves well in several ways, including that it:

- Does not crash if the **user receives a phone call or switches to another app** while using your app
- **Does not consume valuable system resources** when the user is not actively using it
- **Does not lose the user's progress** if they leave your app and return to it at a later time
- **Does not crash or lose the user's progress when the screen rotates** between landscape and portrait orientation

Understand the Lifecycle Callbacks

Resumed: In this state, the **activity is in the foreground and the user can interact with it**. (Also sometimes referred to as the "running" state.)

Paused : In this state, **the activity is partially obscured by another activity**—the other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. **The paused activity does not receive user input and cannot execute any code**

Stopped: In this state, **the activity is completely hidden and not visible to the user**; it is considered to be in the background. While stopped, the activity instance and all its state information such as member variables is retained, but it cannot execute any code

Specify Your App's Launcher Activity

When the **user selects your app icon from the Home screen**, the system **calls the onCreate() method for the Activity in your app** that you've declared **to be the "launcher" (or "main") activity**. This is the activity that serves as the **main entry point to your app's user interface**

```
<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>
```

If either the MAIN action or LAUNCHER category are not declared for one of your activities, ***then your app icon will not appear in the Home screen's list of apps***

Create a New Instance

```
TextView mTextView; // Member variable for text view in the layout

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project res/layout/main_activity.xml file
    setContentView(R.layout.main_activity);

    // Initialize member TextView so we can manipulate it later
    mTextView = (TextView) findViewById(R.id.text_message);

    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // For the main activity, make sure the app icon in the action bar
        // does not behave as a button
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```

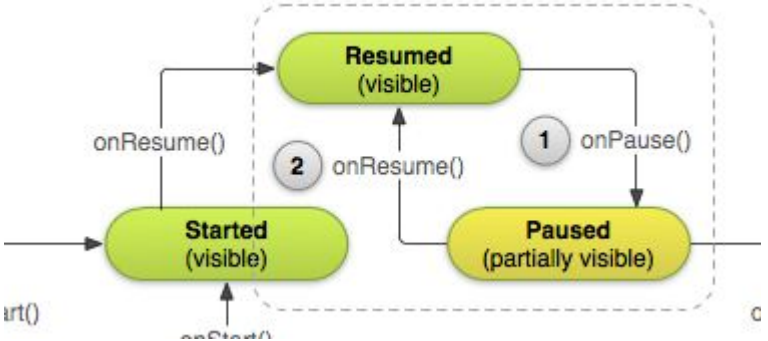
Note!

Once the **onCreate()** finishes execution, the system calls the **onStart() and onResume() methods in quick succession**. Your activity never resides in the Created or Started states. **Technically, the activity becomes visible to the user when onStart() is called, but onResume() quickly follows and the activity remains in the Resumed state until something occurs to change that, such as when a phone call is received, the user navigates to another activity, or the device screen turns off**

Pausing and Resuming an Activity

During normal app use, **the foreground activity is sometimes obstructed by other visual components that cause the activity to pause**. For example, when a semi-transparent activity opens (such as one in the style of a ***dialog***), the previous activity pauses. As long as the activity is still partially visible but currently not the activity in focus, it remains paused

However, **once the activity is fully-obstructed and not visible, it stops**



Pause Your Activity

When the system calls onPause() for your activity, **it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity and it will soon enter the Stopped state**. You should usually use the onPause() callback to:

- **Stop animations or other ongoing actions** that could consume CPU
- **Commit unsaved changes**, but only if users expect such changes to be permanently saved when they leave (**such as a draft email**)
- **Release system resources**, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them

Pause Your Activity

```
@Override
public void onPause() {
    super.onPause();  // Always call the superclass method first

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

Pause Your Activity

Generally, **you should not use onPause() to store user changes** (such as personal information entered into a form) **to permanent storage**. **The only time you should persist user changes** to permanent storage within onPause() is **when you're certain users expect the changes to be auto-saved** (such as when drafting an email).

However, **you should avoid performing CPU-intensive work during onPause()**, such as **writing to a database**, because it can slow the visible transition to the next activity (**you should instead perform heavy-load shutdown operations during onStop()**)

You should keep the amount of operations done in the onPause() method relatively simple in order to allow for a speedy transition to the user's next destination if your activity is actually being stopped

Resume Your Activity

When the user resumes your activity from the Paused state, the system calls the onResume() method

Be aware that the system calls this method every time your activity comes into the foreground, including when it's created for the first time. As such, **you should implement onResume() to initialize components that you release during onPause() and perform any other initializations that must occur each time the activity enters the Resumed state** (such as begin animations and initialize components only used while the activity has user focus)

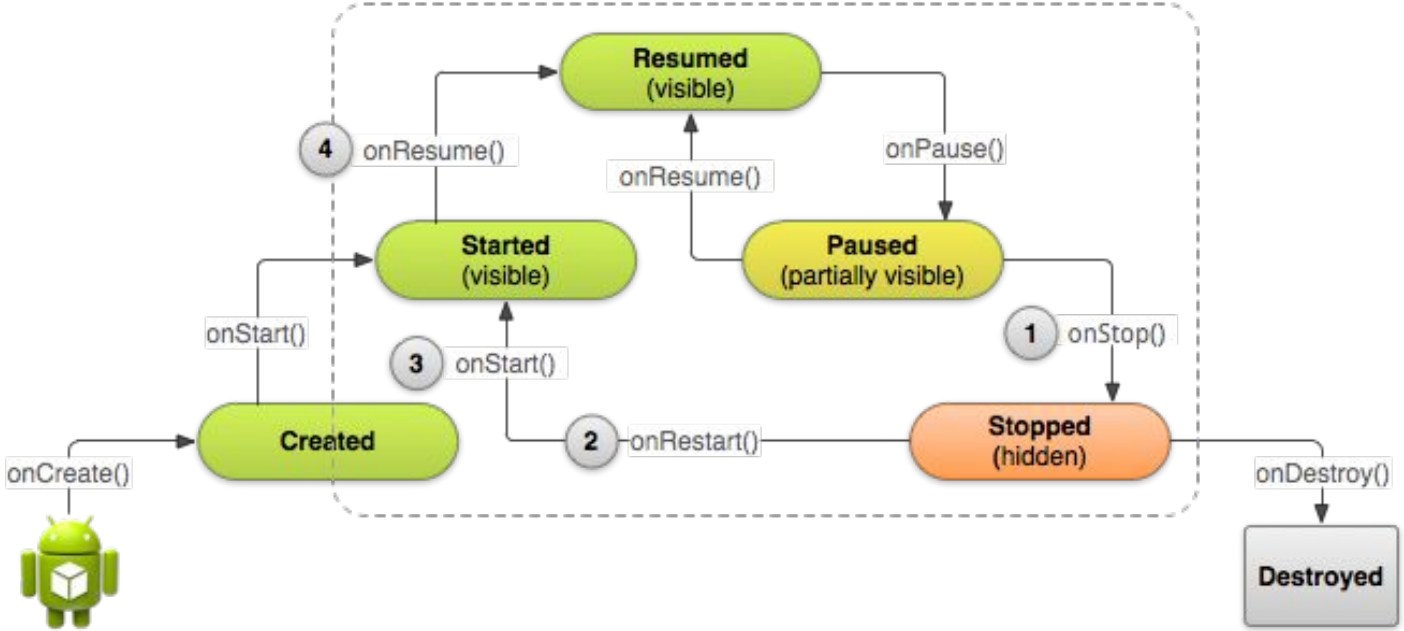
Resume Your Activity

```
@Override
public void onResume() {
    super.onResume();  // Always call the superclass method first

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

Stopping and Restarting an Activity

- The user opens the Recent Apps window and switches from your app to another app. The activity in your app that's currently in the foreground is stopped. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity restarts
- The user performs an action in your app that starts a new activity. The current activity is stopped when the second activity is created. If the user then presses the Back button, the first activity is restarted
- The user receives a phone call while using your app on his or her phone

Stopping and Restarting an Activity



Note

Because the system retains your Activity instance in system memory when it is stopped, it's possible that you don't need to implement the onStop() and onRestart() (or even onStart() methods at all. For most activities that are relatively simple, the activity will stop and restart just fine and you might only need to use onPause() to pause ongoing actions and disconnect from system resources.

Stop Your Activity

Although the onPause() method is called before onStop(), **you should use onStop() to perform larger, more CPU intensive shut-down operations**, such as writing information to a database

```
@Override
protected void onStop() {
    super.onStop();  // Always call the superclass method first

    // Save the note's current draft, because the activity is stopping
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());

    getContentResolver().update(
            mUri,    // The URI for the note to update.
            values,  // The map of column names and new values to apply to them.
            null,    // No SELECT criteria are used.
            null     // No WHERE columns are used.
            );
}
```

Start/Restart Your Activity

When your activity comes back to the foreground from the stopped state, it receives a call to onRestart(). The system also calls the onStart() method, which happens every time your activity becomes visible (whether being restarted or created for the first time)

```
@Override
protected void onStart() {
    super.onStart();  // Always call the superclass method first

    // The activity is either being restarted or started for the first time
    // so this is where we should make sure that GPS is enabled
    LocationManager locationManager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    boolean gpsEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)

    if (!gpsEnabled) {
        // Create a dialog here that requests the user to enable GPS, and use an intent
        // with the android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS action
        // to take the user to the Settings screen to enable GPS when they click "OK"
    }
}
```

Recreating an Activity

There are a few scenarios in which your **activity is destroyed due to normal app behavior**, such as **when the user presses the Back button or your activity signals its own destruction by calling finish()**. The system may also destroy your activity **if it's currently stopped and hasn't been used in a long time or the foreground activity requires more resources so the system must shut down background processes to recover memory**

Recreating an Activity

When your activity is destroyed because the **user presses Back or the activity finishes itself**, the system's concept of that **Activity instance is gone forever** because the behavior indicates the activity is no longer needed.

However, if the system destroys the activity due to **system constraints** (rather than normal app behavior), then **although the actual Activity instance is gone, the system remembers that it existed** such that if the user navigates back to it, the s**ystem creates a new instance of the activity using a set of saved data that describes the state of the activity when it was destroyed**. The saved data that the system uses to restore the previous state is called the "instance state" and is a **collection of key-value pairs stored in a Bundle object**

Caution

Your activity will be destroyed and recreated **each time the user rotates the screen**. When the screen changes orientation, the system destroys and recreates the foreground activity because the screen configuration has changed and your **activity might need to load alternative resources** (such as the layout)

Caution

By default, the system **uses the Bundle instance state to save information about each View object in your activity layout** (such as the text value entered into an EditText object). So, **if your activity instance is destroyed and recreated, the state of the layout is restored to its previous state with no code required by you**. However, your activity might have more state information that you'd like to restore, such as member variables that track the user's progress in the activity

In order for the Android system **to restore the state of the views in your activity**, each view must have a unique ID, supplied by the **android:id** attribute

Save Activity State

To save additional data about the activity state, you must override the **onSaveInstanceState()** callback method. **The system calls this method when the user is leaving your activity and passes it the Bundle object that will be saved in the event that your activity is destroyed unexpectedly**. If the system must r**ecreate the activity instance later, it passes the same Bundle object to both the onRestoreInstanceState() and onCreate() methods**

Save Activity State

Save Your Activity State

Caution: Always call the superclass implementation of onSaveInstanceState() so the default implementation can save the state of the view hierarchy.

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

Restore Your Activity State

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
```

Restore Your Activity State

Instead of restoring the state during onCreate() **you may choose to implement onRestoreInstanceState()**, which the system calls after the onStart() method

```
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

Controlling configuration

An activity is restarted if a configuration change occurs. **A configuration change happens if an event is triggered from the actual the Android device which may be relevant for the application**

An instance of the Configuration class defines the current configuration of the device. Typical configuration is the device **orientation, etc**.

Controlling configuration

For example if the user changes the orientation of the device (vertically or horizontally). **Android assumes that an activity might want to use different resources for these orientations and restarts the activity**

In case an activity is restarted the programmer must ensure that the activity is recreated in the same state as before the restart. The Android provides several potential means for doing this

Controlling configuration

You can avoid a restart of your application for certain configuration changes via the configChanges attribute on your activity definition in your AndroidManifest.xml. The following setting avoids an activity restart incase of orientation changes or position of the physical keyboard (hidden / visible)

```
<activity android:name=".ProgressTestActivity"
     android:label="@string/app_name"
     android:configChanges="orientation|keyboardHidden|keyboard">
</activity>
```

Create your activities

```
public class MainActivity extends TracerActivity {

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

  public void onClick(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    startActivity(intent);
  }
```

```
public class SecondActivity extends       Activity {

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

Building a Dynamic UI with Fragments

To create a **dynamic and multi-pane user interface** on Android, **you need to encapsulate UI components and activity behaviors into modules that you can swap into and out of your activities**

You can create these modules with the **Fragment** class, which behaves somewhat **like a nested activity that can define its own layout and manage its own lifecycle**

You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities)

Creating a Fragment

One difference when creating a Fragment is that you must use the **onCreateView()** callback to define the layout. In fact, this is the only callback you need in order to get a fragment running

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

Add a Fragment to an Activity using XML

res/layout-large/news_articles.xml

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

    <fragment android:name="com.example.android.fragments.HeadlinesFragment"
              android:id="@+id/headlines_fragment"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFragment"
              android:id="@+id/article_fragment"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

</LinearLayout>
```

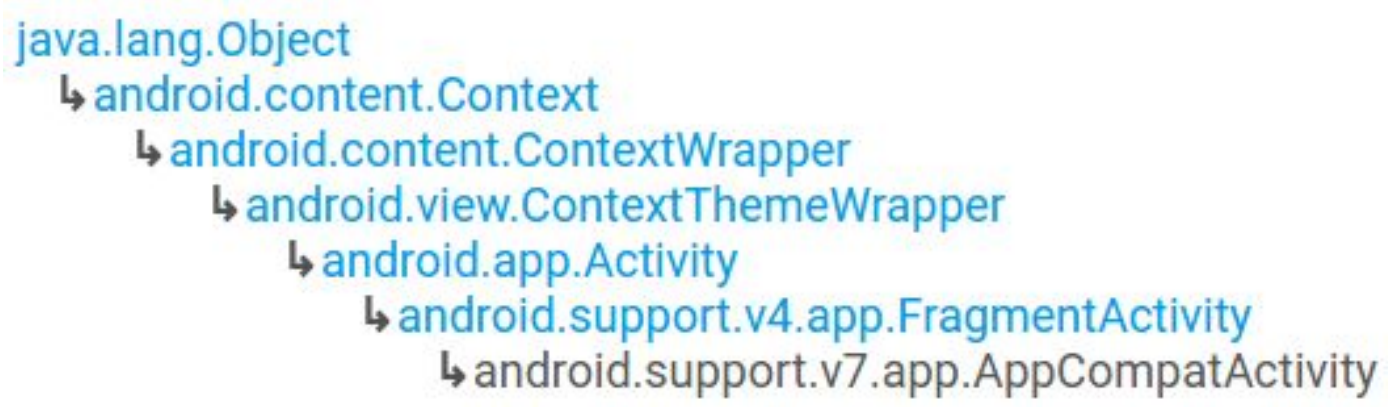
Creating a Fragment

Just like an activity, **a fragment should implement other lifecycle callbacks that allow you to manage its state as it is added or removed from the activity and as the activity transitions between its lifecycle states**

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }
}
```

Creating a Fragment

Just like an activity, **a fragment should implement other lifecycle callbacks that allow you to manage its state as it is added or removed from the activity and as the activity transitions between its lifecycle states**



Note

- **v4 Support Library**: This library is designed to be used with Android 1.6 (API level 4) and higher
- **v7 Support Library**: There are several libraries designed to be used with Android 2.1 (API level 7) and higher
- **v7 appcompat library**: This library adds support for the Action Bar user interface design pattern. This library includes support for material design user interface implementations. This library depends on the v4 Support Library
- **v13 Support Library**: This library is designed to be used for Android 3.2 (API level 13) and higher
- For more information: http://developer.android.com/tools/support-library/features.html

Example

The **toolbar** bar (formerly known as **action bar**) is represented as of Android 5.0 via the Toolbar view group and can be placed into your layout file. It can display the activity title, icon, actions which can be triggered, additional views and other interactive items. It can also be used for navigation in your application

The toolbar has been introduced in Android 5.0 (API 21). If you want to use the toolbar on devices with an earlier Android release you can use the downport provided by the **appcompat-v7 support library**

Supporting Different Platform Versions

The AndroidManifest.xml file describes details about your app and identifies which versions of Android it supports. Specifically, the minSdkVersion and targetSdkVersion attributes for the <uses-sdk element identify the lowest API level with which your app is compatible and the highest API level against which you've designed and tested your app

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

Check System Version at Runtime

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Building a Flexible UI

When designing your application to support a wide range of screen sizes, you can reuse your fragments in different layout configurations to optimize the user experience based on the available screen space

Add a Fragment to an Activity at Runtime

To perform a transaction such as **add or remove a fragment**, you must use the **FragmentManager** to create a **FragmentTransaction**, which provides APIs to **add, remove, replace**, and perform other fragment transactions

Because the fragment has been added to the FrameLayout container at runtime—instead of defining it in the activity's layout with a <fragment> element—the activity can remove the fragment and replace it with a different one

```java
public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);

        // Check that the activity is using the layout version with
        // the fragment_container FrameLayout
        if (findViewById(R.id.fragment_container) != null) {

            // However, if we're being restored from a previous state,
            // then we don't need to do anything and should return or else
            // we could end up with overlapping fragments.
            if (savedInstanceState != null) {
                return;
            }

            // Create a new Fragment to be placed in the activity layout
            HeadlinesFragment firstFragment = new HeadlinesFragment();

            // In case this activity was started with special instructions from an
            // Intent, pass the Intent's extras to the fragment as arguments
            firstFragment.setArguments(getIntent().getExtras());

            // Add the fragment to the 'fragment_container' FrameLayout
            getSupportFragmentManager().beginTransaction()
                    .add(R.id.fragment_container, firstFragment).commit();
        }
    }
}
```

```xml
<FrameLayout xmlns:android="http://schemas.andro
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Replace One Fragment with Another

Keep in mind that when you perform fragment transactions, such as replace or remove one, it's often appropriate to allow the user to navigate backward and "undo" the change. To allow the user to navigate backward through the fragment transactions, you must call **addToBackStack()** before you commit the FragmentTransaction

```java
// Create fragment and give it an argument specifying the article it should show
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack so the user can navigate back
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

Communicating with Other Fragments

In order to reuse the Fragment UI components, **you should build each as a completely self-contained, modular component that defines its own layout and behavior**. Once you have defined these reusable Fragments, you can associate them with an Activity and connect them with the application logic to realize the overall composite UI

Define an Interface

```
public class HeadlinesFragment extends ListFragment {
    OnHeadlineSelectedListener mCallback;

    // Container Activity must implement this interface
    public interface OnHeadlineSelectedListener {
        public void onArticleSelected(int position);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        // This makes sure that the container activity has implemented
        // the callback interface. If not, it throws an exception
        try {
            mCallback = (OnHeadlineSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                    + " must implement OnHeadlineSelectedListener");
        }
    }

    ...
}
```

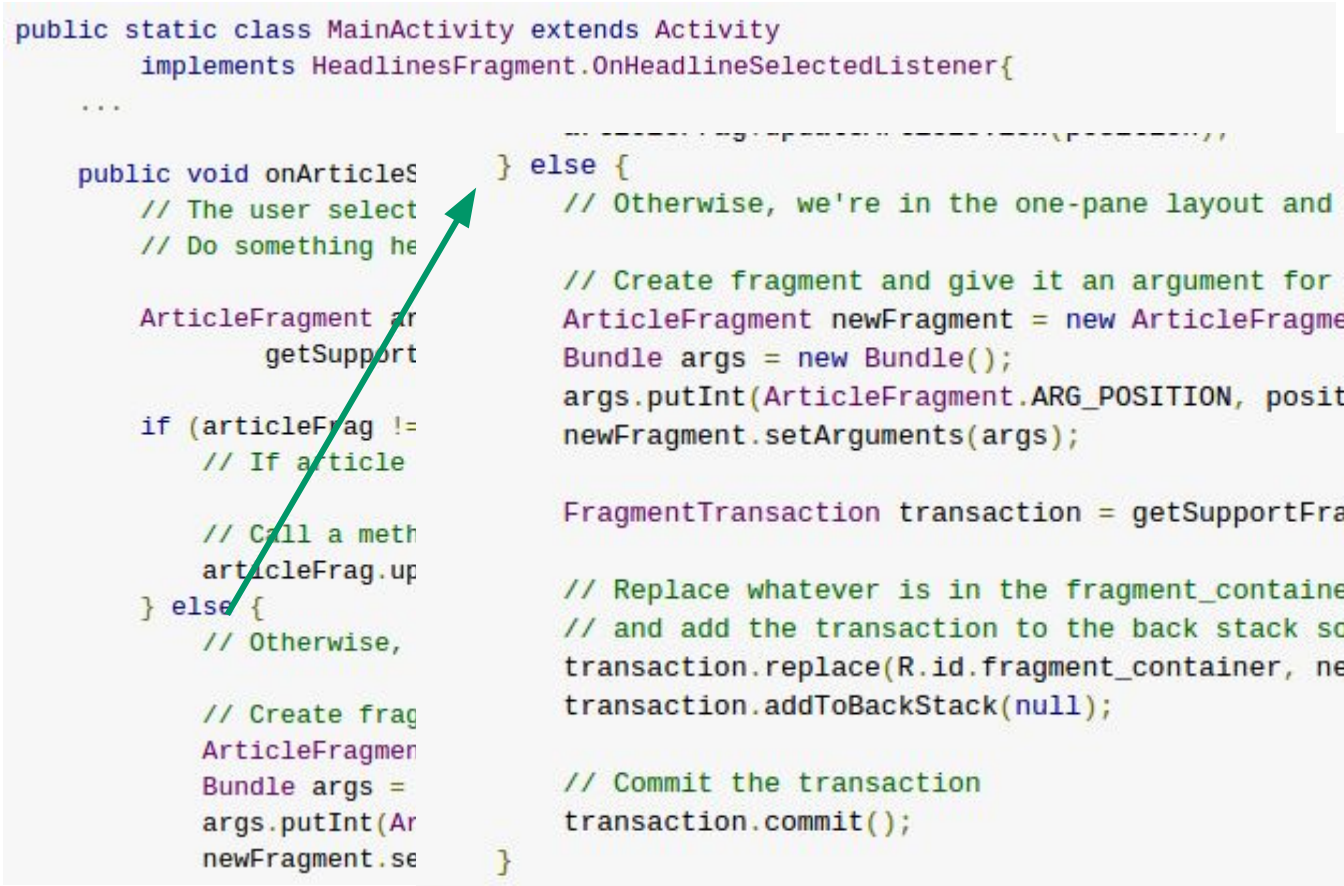
clicks on a list item

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Send the event to the host activity
    mCallback.onArticleSelected(position);
}
```

Implement the Interface

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article
    }
}
```

```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleSelected(int position) {
        // The user selected the headline of an article from the HeadlinesFragment
        // Do something here to display that article

        ArticleFragment articleFrag = (ArticleFragment)
                getSupportFragmentManager().findFragmentById(R.id.article_fragment);

        if (articleFrag != null) {
            // If article frag is available, we're in two-pane layout...

            // Call a method in the ArticleFragment to update its content
            articleFrag.updateArticleView(position);
        } else {
            // Otherwise, we're in the one-pane layout and must swap frags...

            // Create fragment and give it an argument for the selected article
            ArticleFragment newFragment = new ArticleFragment();
            Bundle args = new Bundle();
            args.putInt(ArticleFragment.ARG_POSITION, position);
            newFragment.setArguments(args);
```



```
public static class MainActivity extends Activity
        implements HeadlinesFragment.OnHeadlineSelectedListener{
    ...

    public void onArticleS
        // The user select
        // Do something he

        ArticleFragment ar
                getSupport

        if (articleFrag !=
            // If article

            // Call a meth
            articleFrag.up
        } else {
            // Otherwise,

            // Create frag
            ArticleFragmen
            Bundle args =
            args.putInt(Ar
            newFragment.se
    } else {
        // Otherwise, we're in the one-pane layout and must swap frags...

        // Create fragment and give it an argument for the selected article
        ArticleFragment newFragment = new ArticleFragment();
        Bundle args = new Bundle();
        args.putInt(ArticleFragment.ARG_POSITION, position);
        newFragment.setArguments(args);

        FragmentTransaction transaction = getSupportFragmentManager().beginTransaction()

        // Replace whatever is in the fragment_container view with this fragment,
        // and add the transaction to the back stack so the user can navigate back
        transaction.replace(R.id.fragment_container, newFragment);
        transaction.addToBackStack(null);

        // Commit the transaction
        transaction.commit();
    }
}
```

Headless fragments

Headless Fragments are Fragments which do not have any UI

Headless Fragments, have one really useful feature - they can be retained by the FragmentManager across configuration changes

For simpler objects this is an overkill and I recommend just keeping the state by implementing the onSaveInstanceState and onRestoreInstance methods. However, not everything can be implemented in a way that it can fit into a Bundle

Headless fragments

Some classes are just **too complicated to be implemented** as a Parceable. Sometimes one simply **does not want to write all the extra boilerplate code needed for it**. Or sometimes **one wants a simple AsyncTask to do something and keep doing it across orientation change**

- For more information: *http://developer.android.com/guide/topics/resources/runtime-changes.html#HandlingTheChange*

Supporting Different Languages

It’s always a good practice to **extract UI strings from your app code and keep them in an external file**. Android makes this easy with a **resources directory** in each Android project

```
MyProject/
    res/
        values/
            strings.xml
        values-es/
            strings.xml
        values-fr/
            strings.xml
```

/values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

/values-es/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Aplicación</string>
    <string name="hello_world">Hola Mundo!</string>
</resources>
```

/values-fr/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Use the String Resources

You can reference your string resources in your source code and other XML files using the resource name defined by the <string> element's name attribute. In your source code, you can refer to a string resource with the syntax **R.string.<string_name>**. There are a variety of methods that accept a string resource this way

```
// Get a string resource from your app's  Resources
String hello =  getResources() .getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Supporting Different Screens

Android categorizes device screens using two general properties: **size and density**

There are four generalized sizes: **small, normal, large, xlarge**

And four generalized densities: **low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)**

Supporting Different Screens

The system loads the layout file from the appropriate layout directory based on screen size of the device on which your app is running

```
MyProject/
    res/
        layout/             # default (portrait)
            main.xml
        layout-land/        # landscape
            main.xml
        layout-large/       # large (portrait)
            main.xml
        layout-large-land/  # large landscape
            main.xml
```

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

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

Terms and concepts

Screen size: **Actual physical size**, measured as the screen's diagonal.

For simplicity, Android groups all actual screen sizes into four generalized sizes: **small, normal, large, and extra-large**

Screen density: **The quantity of pixels within a physical area of the screen;** usually referred to as **dpi (dots per inch)**. For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen.

For simplicity, Android groups all actual screen densities into six generalized densities: **low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high**

Terms and concepts

Orientation: The **orientation of the screen** from the user's point of view. This is either **landscape or portrait**, meaning that the screen's aspect ratio is either wide or tall, respectively. Be aware that not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device

Resolution: The **total number of physical pixels on a screen**. When adding support for multiple screens, ***applications do not work directly with resolution***; ***applications should be concerned only with screen size and density,*** as specified by the generalized size and density groups

Terms and concepts

Density-independent pixel (dp): **A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way**

The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen. At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple: **px = dp * (dpi / 160)**. For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities

Converting dp units to pixel units

n some cases, you will need to express dimensions in dp and then convert them to pixels. Imagine an application in which a scroll or fling gesture is recognized after the user's finger has moved by at least 16 pixels. On a baseline screen, a user's must move by 16 pixels / 160 dpi, which equals 1/10th of an inch (or 2.5 mm) before the gesture is recognized. On a device with a high-density display (240dpi), the user's must move by 16 pixels / 240 dpi, which equals 1/15th of an inch (or 1.7 mm). The distance is much shorter and the application thus appears more sensitive to the user.

To fix this issue, the gesture threshold must be expressed in code in dp and then converted to actual pixels. For example:

Converting dp units to pixel units

```
// The gesture threshold expressed in dp
private static final float GESTURE_THRESHOLD_DP = 16.0f;

// Get the screen's density scale
final float scale =  getResources() . getDisplayMetrics() .density;
// Convert the dps to pixels, based on density scale
mGestureThreshold = (int) (GESTURE_THRESHOLD_DP * scale + 0.5f);

// Use mGestureThreshold as a distance in pixels...
```

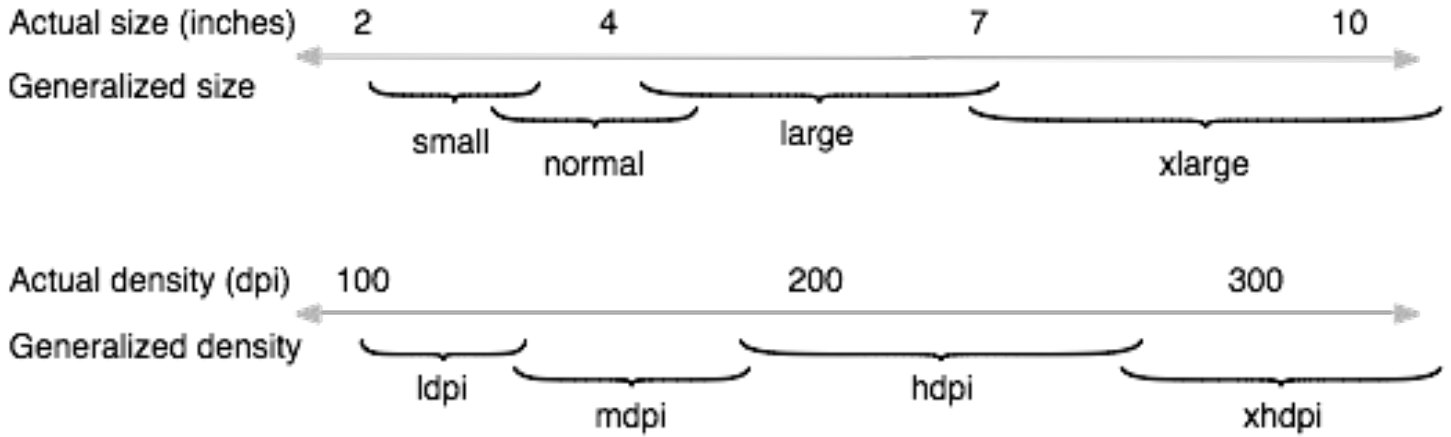
Terms and concepts

A set of six generalized densities

- ldpi (low) ~120dpi
- mdpi (medium) ~160dpi
- hdpi (high) ~240dpi
- xhdpi (extra-high) ~320dpi
- xxhdpi (extra-extra-high) ~480dpi
- xxxhdpi (extra-extra-extra-high) ~640dpi

Range of screens supported

To simplify the way that you design your user interfaces for multiple screens, Android divides the range of actual screen sizes and densities into



Range of screens supported

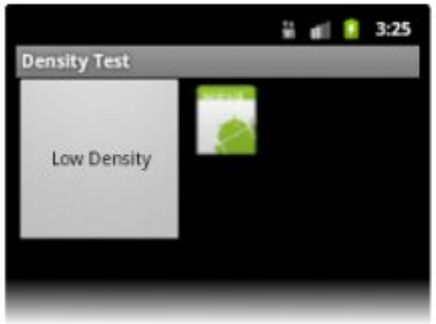
Each generalized size and density spans a range of actual screen sizes and densities. **For example, two devices that both report a screen size of normal might have actual screen sizes and aspect ratios that are slightly different when measured by hand**. Similarly, two devices that report a screen density of hdpi might have real pixel densities that are slightly different. Android makes these differences abstract to applications, so you can provide UI designed for the generalized sizes and densities and let the system handle any final adjustments as necessary

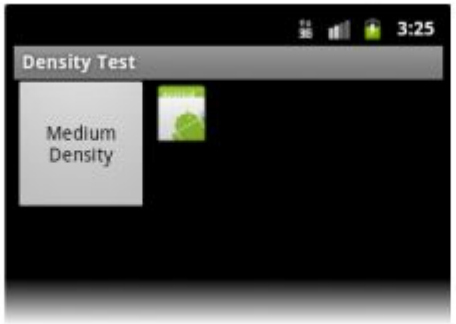
Note

These minimum screen sizes were not as well defined prior to Android 3.0, so **you may encounter some devices that are mis-classified between normal and large**. These are also based on the physical resolution of the screen, so may vary across devices—for example a 1024x720 tablet with a system bar actually has a bit less space available to the application due to it being used by the system bar

Density independence

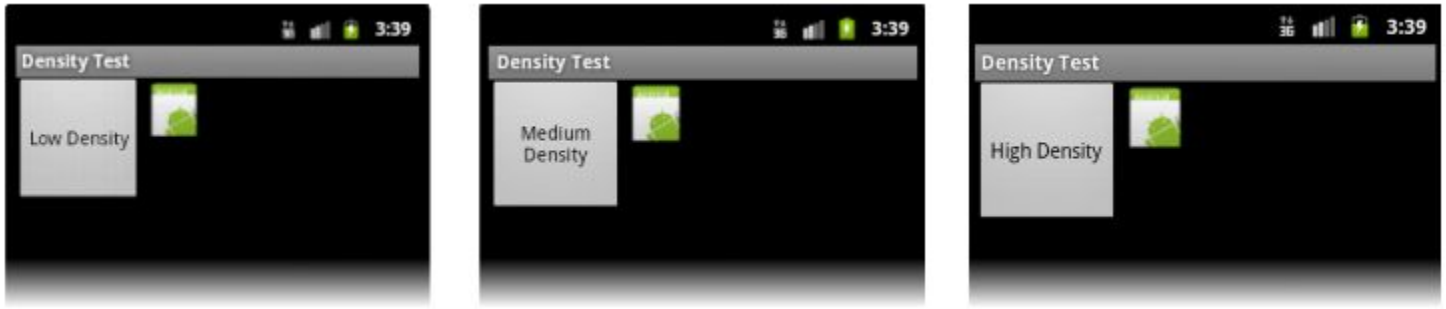
Example application without support for different densities, as shown on low, medium, and high-density screens





Density independence

Example application with good support for different densities (it's density independent), as shown on low, medium, and high density screens



Density independence

In most cases, you can ensure density independence in your application simply by specifying all layout dimension values in density-independent pixels (**dp units**) or with "**wrap_content**", as appropriate. The system then scales bitmap drawables as appropriate in order to display at the appropriate size, based on the appropriate scaling factor for the current screen's density

However, bitmap scaling can result in blurry or pixelated bitmaps, which you might notice in the above screenshots. **To avoid these artifacts, you should provide alternative bitmap resources for different densities**

Create Different Bitmaps

You should always provide bitmap resources that are properly scaled to each of the generalized density buckets: low, medium, high and extra-high density

To generate these images, you should start with your r**aw resource in vector format and generate the images for each density** using the following size scale

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0 (baseline)
- ldpi: 0.75

This means that if you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices

Create Different Bitmaps

Any time you reference @drawable/awesomeimage, the system selects the appropriate bitmap based on the screen's density

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Low-density (ldpi) resources aren’t always necessary. When you provide hdpi assets, the system scales them down by one half to properly fit ldpi screens

Important resource qualifiers

Using density as resource qualifier

Using orientation as resource qualifier

Android version qualifiers: Another typical selection is the smallest available width selection or the available width selection. The width selection can, for example, be used to provide different layouts based on the width of the device. This selection is based on -**sw[Number]dp (Smallest) or -w[Number]dp qualifier, where [Number] stands for the number of device independent pixels**. For example, a 7inch tablet typically has at least 600dp and you could provide layouts for it via the **res/layout-sw600dp/** selector

Smallest possible width

sw<N>dp: The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (**you may also think of it as the "smallest possible width" for the screen**). You can use this qualifier to ensure that, regardless of the screen's current orientation, your application's has at least <N> dps of width available for its UI

```
res/layout/main_activity.xml           # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml   # For 7” tablets (600dp wide and bigger)
res/layout-sw720dp/main_activity.xml   # For 10” tablets (720dp wide and bigger)
```

Available screen width and height

h<N>dp: Examples: h720dp, h1024dp

w<N>dp: Examples: w720dp, w1024dp

Specifies a minimum screen width/height in dp units at which the resources should be used—defined by the <N> value. The system's corresponding value for the width/height changes when the screen's orientation switches between landscape and portrait to reflect the current actual height that's available for your UI.

```
res/layout/main_activity.xml          # For handsets (smaller than 600dp available width)
res/layout-w600dp/main_activity.xml   # Multi-pane (any screen with 600dp available width or more)
```

Defining the size of UI components in layout files

Fixed or relative dimensions: Android allows you to define the size of user interface components in fixed or relative dimensions in the layout files

Using dp as relative dimension: The unit of measurement which should be used is dp. **dp is short for dip (device independent pixel)**

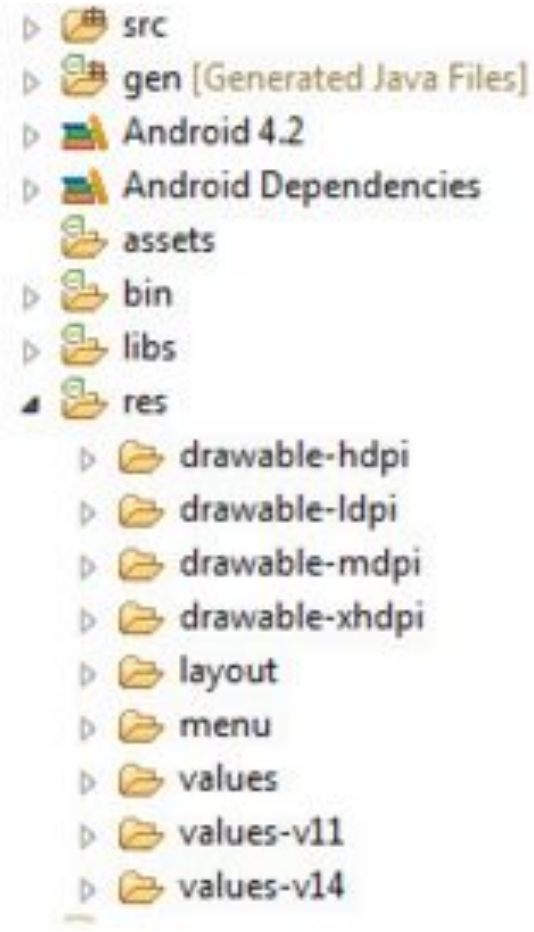
Using sp to scale with the user text preferences

If the unit should scale with text preference settings of the user, **choose the sp unit of measurement**. **This unit is similar to dp, but it is also scaled by the user preference**

If the users select to increase the font size in this settings, views which use sp are scaled accordingly

Resource Types

- Animation Resources
- Color State List Resource
- Drawable Resources
- Layout Resource
- Menu Resource
- String Resources
- Style Resource
- More Resource Types



Animation Resources

Define pre-determined animations

Tween animations are saved in **res/anim/** and accessed from the **R.anim** class

Frame animations are saved in **res/drawable/** and accessed from the **R.drawable** class.

Animation Resources

An animation resource can define one of two types of animations:

- **Property Animation**: Creates an animation by **modifying an object's property values over a set period of time with an Animator**
- **View Animation**: There are two types of animations that you can do with the view animation framework:
  - **Tween animation**: Creates an animation by **performing a series of transformations on a single image with an Animation**
  - **Frame animation**: or creates an animation **by showing a sequence of images in order with an AnimationDrawable**

Property Animation

An animation defined in XML that modifies properties of the target object, such as background color or alpha value, over a set amount of time

FILE LOCATION:

- res/anim/filename.xml
- The filename will be used as the resource ID

COMPILED RESOURCE DATATYPE:

- Resource pointer to a **ValueAnimator**, **ObjectAnimator**, or **AnimatorSet**

RESOURCE REFERENCE:

- In Java: **R.anim.filename**
- In XML: **@[package:]anim/filename**

Property Animation

The file must have a single root element: either <set>, <objectAnimator>, or <valueAnimator>

```
<set
  android:ordering=["together" | "sequentially"]>

    <objectAnimator
        android:propertyName="string"
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <animator
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <set>
        ...
    </set>
</set>
```

<set>

- **A container that holds other animation elements** (<objectAnimator>, <valueAnimator>, or other <set> elements

- Android:ordering
  - Sequentially: Play animations in this set sequentially
  - together (default): Play animations in this set at the same time

<objectAnimator>

Animates a specific property of an object over a specific amount of time

- **android:propertyName**: String. Required. The object's property to animate, referenced by its name. For example you can specify "alpha" or "backgroundColor" for a View object
- **android:valueTo**: float, int, or color. Required. The value where the animated property ends. Colors are represented as six digit hexadecimal numbers (for example, #333333)
- **android:valueFrom**: float, int, or color. The value where the animated property starts
- **android:duration**:int. The time in milliseconds of the animation. 300 milliseconds is the default

<objectAnimator>

- **android:startOffset**: int. The amount of milliseconds the animation delays after start() is called
- **android:repeatCount**: int. How many times to repeat an animation. Set to "-1" to infinitely repeat or to a positive integer. For example, a value of "1" means that the animation is repeated once after the initial run of the animation, so the animation plays a total of two times. The default value is "0", which means no repetition
- **android:repeatMode**: int (-1 or >0). How an animation behaves when it reaches the end of the animation. Set to "reverse" to have the animation reverse direction with each iteration or "repeat" to have the animation loop from the beginning each time

<objectAnimator>

- **android:valueType**: Keyword. Do not specify this attribute if the value is a color. The animation framework automatically handles color values
  - intType: Specifies that the animated values are integers
  - floatType (default): Specifies that the animated values are floats

<animator>

Performs an animation over a specified amount of time

- android:valueTo
- android:valueFrom
- android:duration
- android:startOffset
- android:repeatCount
- android:repeatMode
- android:valueType

EXAMPLE

XML file saved at
res/animator/property_animator.xml

```xml
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext,
    R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

View Animation - Tween animation

An animation defined in XML that performs transitions such as **rotating, fading, moving, and stretching** on a graphic

FILE LOCATION:

- **res/anim/filename.xml**
- The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

- Resource pointer to an Animation.

RESOURCE REFERENCE:

- In Java: **R.anim.filename**
- In XML: **@[package:]anim/filename**

Syntax

The file must have a single root element: either an **<alpha>, <scale>, <translate>, <rotate>, or <set>** element that holds a group (or groups) of other animation elements (even nested <set> elements).

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"] >
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
        android:pivotX="float"
        android:pivotY="float" />
    <translate
        android:fromXDelta="float"
        android:toXDelta="float"
        android:fromYDelta="float"
        android:toYDelta="float" />
    <rotate
        android:fromDegrees="float"
        android:toDegrees="float"
        android:pivotX="float"
        android:pivotY="float" />
    <set>
        ...
    </set>
</set>
```

<set>

A container that holds other animation elements (<alpha>, <scale>, <translate>, <rotate>) or other <set> elements

- **android:interpolator**: Interpolator resource. An Interpolator to apply on the animation. The value must be a reference to a resource that specifies an interpolator (not an interpolator class name). There are default interpolator resources available from the platform or you can create your own interpolator resource
- android:shareInterpolator: Boolean. "true" if you want to share the same interpolator among all child elements

<alpha>

A fade-in or fade-out animation. Represents an AlphaAnimation

- **android:fromAlpha**: Float. Starting opacity offset, where 0.0 is transparent and 1.0 is opaque.
- **android:toAlpha**: Float. Ending opacity offset, where 0.0 is transparent and 1.0 is opaque.

<scale>

A resizing animation. You can specify the center point of the image from which it grows outward (or inward) by specifying pivotX and pivotY. For example, if these values are 0, 0 (top-left corner), all growth will be down and to the right. Represents a **ScaleAnimation**

<scale>

- **android:fromXScale**: Float. Starting X size offset, where 1.0 is no change.
- **android:toXScale: Float. Ending X size offset, where 1.0 is no change.**
- **android:fromYScale**: Float. Starting Y size offset, where 1.0 is no change.
- **android:toYScale**: Float. Ending Y size offset, where 1.0 is no change.
- **android:pivotX**: Float. The X coordinate to remain fixed when the object **is scaled.**
- **android:pivotY**: Float. The Y coordinate to remain fixed when the object is scaled

<translate>

A vertical and/or horizontal motion. Supports the following attributes in any of the following three formats: **values from -100 to 100 ending with "%", indicating a percentage relative to itself**; values from -100 to 100 ending in "%p", indicating **a percentage relative to its parent**; a float value with no suffix, indicating an absolute value. Represents a **TranslateAnimation**

<translate>

- **android:fromXDelta:** Float or percentage. Starting X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p")

- **android:toXDelta:** Float or percentage. Ending X offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element width (such as "5%"), or in percentage relative to the parent width (such as "5%p")

<translate>

- **android:fromYDelta:** Float or percentage. Starting Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p")

- **android:toYDelta:** Float or percentage. Ending Y offset. Expressed either: in pixels relative to the normal position (such as "5"), in percentage relative to the element height (such as "5%"), or in percentage relative to the parent height (such as "5%p")

<rotate>

A rotation animation. Represents a RotateAnimation

- **android:fromDegrees**: Float. Starting angular position, in degrees.
- **android:toDegrees**: Float. Ending angular position, in degrees.
- **android:pivotX**: Float or percentage. **The X coordinate of the center of rotation**. Expressed either: *in pixels relative to the object's left edge* (such as "5"), *in percentage relative to the object's left edge* (such as "5%"), or in *percentage relative to the parent container's left edge* (such as "5%p").
- **android:pivotY**: Float or percentage. The Y coordinate of the center of rotation. Expressed either: *in pixels relative to the object's top edge* (such as "5"), *in percentage relative to the object's top edge* (such as "5%"), or *in percentage relative to the parent container's top edge* (such as "5%p").

EXAMPLE

XML file saved at res/anim/hyperspace_jump.xml

```xml
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set
        android:interpolator="@android:anim/accelerate_interpolator"
        android:startOffset="700">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
    </set>
```

EXAMPLE

application code will apply the animation to an ImageView and start the animation:

```
ImageView image = (ImageView) findViewById(R.id.image);

Animation hyperspaceJump = AnimationUtils.loadAnimation(this,
R.anim.hyperspace_jump);

image.startAnimation(hyperspaceJump);
```

Interpolators

Interpolator class	Resource ID
AccelerateDecelerateInterpolator	@android:anim/accelerate_decelerate_interpolator
AccelerateInterpolator	@android:anim/accelerate_interpolator
AnticipateInterpolator	@android:anim/anticipate_interpolator
AnticipateOvershootInterpolator	@android:anim/anticipate_overshoot_interpolator
BounceInterpolator	@android:anim/bounce_interpolator
CycleInterpolator	@android:anim/cycle_interpolator
DecelerateInterpolator	@android:anim/decelerate_interpolator
LinearInterpolator	@android:anim/linear_interpolator
OvershootInterpolator	@android:anim/overshoot_interpolator

View Animation - Frame animation

An animation defined in XML that shows a sequence of images in order (like a film)

FILE LOCATION:

- **res/drawable/filename.xml**
- The filename will be used as the resource ID.

COMPILED RESOURCE DATATYPE:

- Resource pointer to an AnimationDrawable.

RESOURCE REFERENCE:

- In Java: **R.drawable.filename**
- In XML: **@[package:]drawable.filename**

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource_name"
        android:duration="integer" />
</animation-list>
```

<animation-list>

Required. This must be the root element. **Contains one or more <item> elements**

- **Android:oneshot**: Boolean. "true" if you want to perform the animation once; "false" to loop the animation

<item>

A single frame of animation. Must be a child of a <animation-list> element

- **Android:drawable**: Drawable resource. The drawable to use for this frame

- **Android:duration**: Integer. The duration to show this frame, in milliseconds

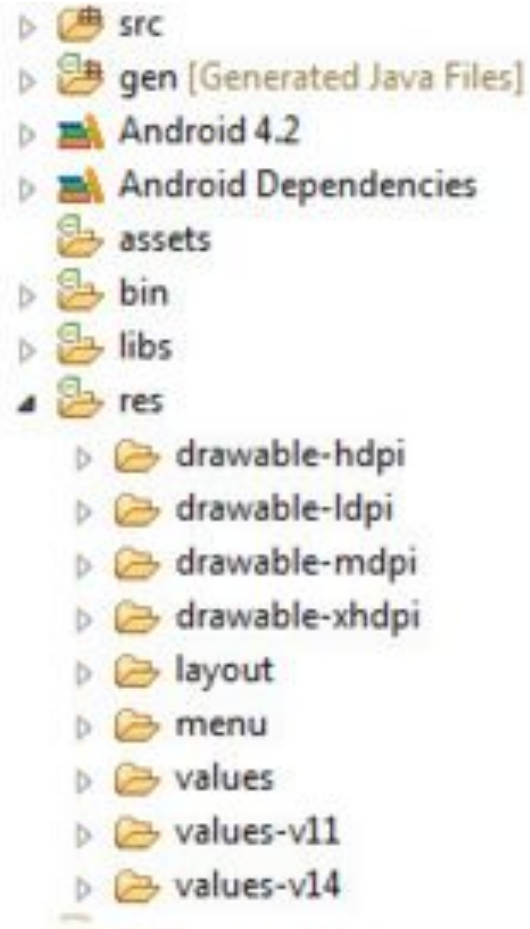
EXAMPLE

```xml
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
android:oneshot="false">
    <item android:drawable="@drawable/rocket_thrust1"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"
android:duration="200" />
</animation-list>
```

```
ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
rocketImage. setBackgroundResource (R.drawable.rocket_thrust);

rocketAnimation = (AnimationDrawable) rocketImage. getBackground() ;
rocketAnimation. start() ;
```

Resource Types

- Animation Resources
- **Color State List Resource**
- Drawable Resources
- Layout Resource
- Menu Resource
- String Resources
- Style Resource
- More Resource Types



Color State List Resource

A **ColorStateList is an object you can define in XML** that you can apply as a color, but will actually **change colors, depending on the state of the View object to which it is applied**. For example, a Button widget can exist in one of several different states (pressed, focused, or neither) and, using a color state list, you can provide a different color during each state

FILE LOCATION:

- **res/color/filename.xml**
- The filename will be used as the resource ID.

RESOURCE REFERENCE:

- In Java: **R.color.filename**
- In XML: **@[package:]color/filename**

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:color="hex_color"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

<selector>

Required. This must be the root element. Contains one or more <item> elements

Xmlns:android : String. Required. Defines the XML namespace, which must be "http://schemas.android.com/apk/res/android"

<item>

Defines a color to use during certain states, as described by its attributes. Must be a child of a <selector> element

android:color: Hexadeximal color. **Required**. The color is specified with an RGB value and optional alpha channel. The value always begins with a pound (#) character and then followed by the **Alpha-Red-Green-Blue** information in one of the following formats:

#RGB
#ARGB
#RRGGBB
#AARRGGBB

<item>

android:state_pressed: Boolean. "true" if this item should be used **when the object is pressed** (such as when a button is touched/clicked); "false" if this item should be used in the default, non-pressed state.

android:state_focused: Boolean. "true" if this item should be used **when the object is focused** (such as when a button is highlighted using the trackball/d-pad); "false" if this item should be used in the default, non-focused state.

<item>

android:state_selected: Boolean. "true" if this item should be used when the **object is selected** (such as when a tab is opened); "false" if this item should be used when the object is not selected.

android:state_checkable: Boolean. "true" if this item should be **used when the object is checkable**; "false" if this item should be used when the object is not checkable. (Only useful if the object can transition between a checkable and non-checkable widget.)

<item>

android:state_checked: Boolean. "true" if this item should be used when the object is checked; "false" if it should be used when the object is un-checked.

android:state_enabled: Boolean. "true" if this item should be used when the object is enabled (capable of receiving touch/click events); "false" if it should be used when the object is disabled.

EXAMPLE

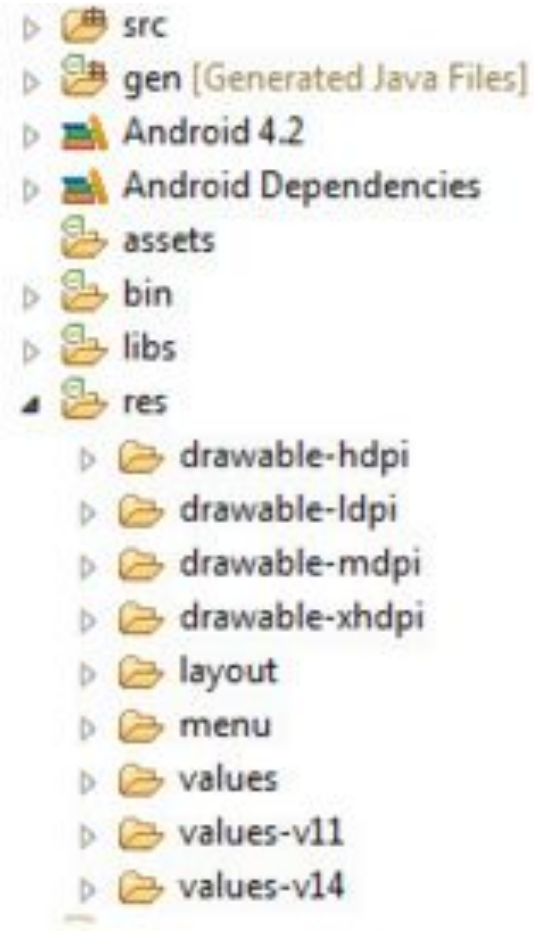
XML file saved at res/color/button_text.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
          android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

```xml
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

Resource Types

- Animation Resources
- Color State List Resource
- **Drawable Resources**
- Layout Resource
- Menu Resource
- String Resources
- Style Resource
- More Resource Types



Drawable Resources

A drawable **resource is a general concept for a graphic that can be drawn to the screen** and which you can retrieve with APIs such as **getDrawable(int)** or apply to another XML resource with attributes such as **android:drawable** and android:icon. There are several different types of drawables:

- **Bitmap File**: A bitmap graphic file (**.png, .jpg, or .gif**). Creates a **BitmapDrawable**.
- **Nine-Patch File**: A PNG file with **stretchable regions to allow image resizing based on content** (**.9.png**). Creates a **NinePatchDrawable**.
- **Layer List**: A Drawable that manages an **array of other Drawables**. **These are drawn in array order**, so the element with the largest index is be drawn on top. Creates a **LayerDrawable**.

Drawable Resources (2)

- **State List**: An XML file that **references different bitmap graphics for different states** (for example, to use a different image when a button is pressed). Creates a **StateListDrawable**.
- **Level List:** An XML file that defines **a drawable that manages a number of alternate Drawables**, each assigned a maximum numerical value. Creates a **LevelListDrawable**.
- **Transition Drawable**: An XML file that defines **a drawable that can cross-fade between two drawable resources**. Creates a **TransitionDrawable**.

Drawable Resources (3)

- **Clip Drawable**: An XML file that defines **a drawable that clips another Drawable based on this Drawable's current level value**. Creates a **ClipDrawable**.
- **Scale Drawable**: An XML file that defines **a drawable that changes the size of another Drawable based on its current level value**. Creates a **ScaleDrawable**
- **Shape Drawable**: An XML file that defines **a geometric shape, including colors and gradients**. Creates a **ShapeDrawable**

Bitmap File

A bitmap file is a **.png, .jpg, or .gif** file. Android creates a Drawable resource for any of these files when you save them in the res/drawable/ directory

FILE LOCATION:

- res/drawable/filename.png (.png, .jpg, or .gif)
- The filename is used as the resource ID.

COMPILED RESOURCE DATATYPE:

- Resource pointer to a **BitmapDrawable**.

RESOURCE REFERENCE:

- In Java: **R.drawable.filename**
- In XML: **@[package:]drawable/filename**

Bitmap File

With an image saved at **res/drawable/myimage.png**

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

```
Resources res =  getResources() ;
Drawable drawable = res. getDrawable (R.drawable.myimage);
```

XML Bitmap

An XML bitmap is a resource defined in XML that points to a bitmap file. The effect is an alias for a raw bitmap file. The XML can specify additional properties for the bitmap such as dithering and tiling

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:antialias=["true" | "false"]
    android:dither=["true" | "false"]
    android:filter=["true" | "false"]
    android:gravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
                     "fill_vertical" | "center_horizontal" | "fill_horizontal" |
                     "center" | "fill" | "clip_vertical" | "clip_horizontal"]
    android:mipMap=["true" | "false"]
    android:tileMode=["disabled" | "clamp" | "repeat" | "mirror"] />
```

Nine-Patch

A NinePatch is **a PNG image in which you can define stretchable regions that Android scales when content within the View exceeds the normal image bounds**. You typically assign this type of image as the background of a View that has at least one dimension set to "wrap_content", and when the View grows to accomodate the content, the Nine-Patch image is also scaled to match the size of the View. An example use of a Nine-Patch image is the background used by Android's standard Button widget, which must stretch to accommodate the text (or image) inside the button.

Nine-Patch

FILE LOCATION:

- **res/drawable/filename.9.png**
- The filename is used as the resource ID.

COMPILED RESOURCE DATATYPE:

- Resource pointer to a **NinePatchDrawable**.

RESOURCE REFERENCE:

- In Java: **R.drawable.filename**
- In XML: **@[package:]drawable/filename**

EXAMPLE: With an image saved at *res/drawable/myninepatch.9.png*

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/myninepatch" />
```

Layer List

A LayerDrawable is a drawable object that manages an array of other drawables. Each drawable in the list is drawn in the order of the list—the last drawable in the list is drawn on top

FILE LOCATION:

- **res/drawable/filename.xml**
- The filename is used as the resource ID.

COMPILED RESOURCE DATATYPE:

- Resource pointer to a **LayerDrawable**.

RESOURCE REFERENCE:

- In Java: **R.drawable.filename**
- In XML: **@[package:]drawable/filename**

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</layer-list>
```

Example

XML file saved at res/drawable/layers.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
      <bitmap android:src="@drawable/android_red"
        android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
      <bitmap android:src="@drawable/android_green"
        android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
      <bitmap android:src="@drawable/android_blue"
        android:gravity="center" />
    </item>
</layer-list>
```

```xml
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/layers" />
```

State List

A StateListDrawable is a drawable object defined in XML that uses a several different images to represent the same graphic, depending on the state of the object. For example, a Button widget can exist in one of several different states (pressed, focused, or neither) and, using a state list drawable, you can provide a different background image for each state

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android"
    android:constantSize=["true" | "false"]
    android:dither=["true" | "false"]
    android:variablePadding=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_hovered=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_activated=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
</selector>
```

Example

XML file saved at res/drawable/button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

Level List

A Drawable that manages a number of alternate Drawables, each assigned a maximum numerical value. Setting the level value of the drawable with setLevel() loads the drawable resource in the level list that has a android:maxLevel value greater than or equal to the value passed to the method

```
<?xml version="1.0" encoding="utf-8"?>
<level-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/drawable_resource"
        android:maxLevel="integer"
        android:minLevel="integer" />
</level-list>
```

Example

Once this is applied to a View, the level can be changed with setLevel() or setImageLevel()

```
<?xml version="1.0" encoding="utf-8"?>
<level-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/status_off"
        android:maxLevel="0" />
    <item
        android:drawable="@drawable/status_on"
        android:maxLevel="1" />
</level-list>
```

Transition Drawable

A TransitionDrawable is a drawable object that can cross-fade between the two drawable resources. Each drawable is represented by an <item> element inside a single <transition> element. **No more than two items are supported**. To transition forward, call **startTransition()**. To transition backward, call **reverseTransition()**.

```
<?xml version="1.0" encoding="utf-8"?>
<transition
xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@[+][package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</transition>
```

EXAMPLE

XML file saved at res/drawable/transition.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

```xml
<ImageButton
    android:id="@+id/button"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/transition" />
```

```
ImageButton button = (ImageButton) findViewById(R.id.button);
TransitionDrawable drawable = (TransitionDrawable) button.getDrawable();
drawable.startTransition(500);
```

Clip Drawable

A drawable defined in XML that clips another drawable based on this Drawable's current level. You can control how much the child drawable gets clipped in width and height based on the level, as well as a gravity to control where it is placed in its overall container. Most often used to implement things like progress bars

```
<?xml version="1.0" encoding="utf-8"?>
<clip
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:clipOrientation=["horizontal" | "vertical"]
    android:gravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
                     "fill_vertical" | "center_horizontal" | "fill_horizontal" |
                     "center" | "fill" | "clip_vertical" | "clip_horizontal"] />
```

EXAMPLE

XML file saved at res/drawable/clip.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/android"
    android:clipOrientation="horizontal"
    android:gravity="left" />
```

```xml
<ImageView
    android:id="@+id/image"
    android:background="@drawable/clip"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

The following code gets the drawable and increases the amount of clipping in order to progressively reveal the image

```
ImageView imageview = (ImageView) findViewById(R.id.image);
ClipDrawable drawable = (ClipDrawable) imageview.getDrawable();
drawable.setLevel(drawable.getLevel() + 1000);
```

Shape Drawable

This is a generic shape defined in XML

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:useLevel=["true" | "false"] />
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <size
        android:width="integer"
        android:height="integer" />
    <solid
        android:color="color" />
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>
```

EXAMPLE

XML file saved at res/drawable/gradient_box.xml

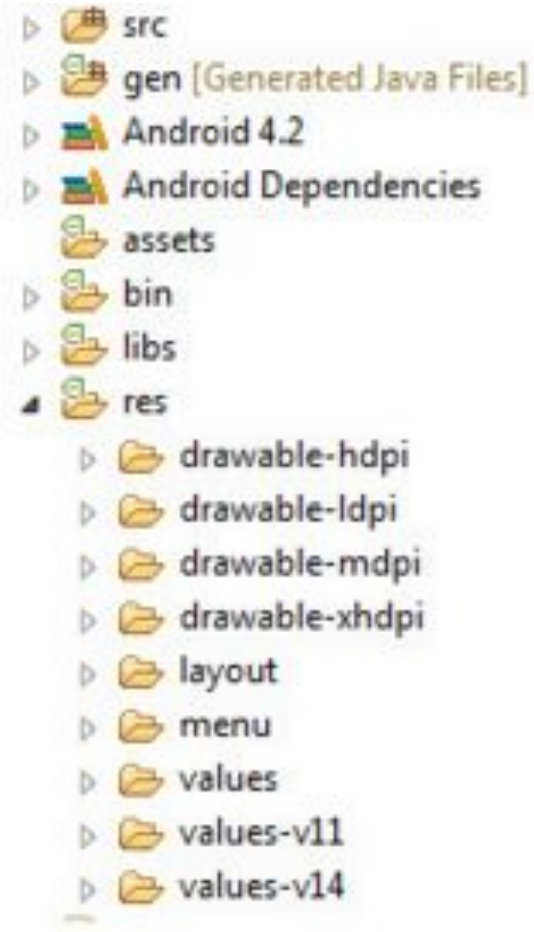
```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
</shape>
```

```
Resources res =  getResources() ;
Drawable shape = res.  getDrawable (R.drawable.gradient_box);

TextView tv = (TextView)findViewByID(R.id.textview);
tv.setBackground(shape);
```

```xml
<TextView
    android:background="@drawable/gradient_box"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

Resource Types

- Animation Resources
- Color State List Resource
- Drawable Resources
- **Layout Resource**
- **Menu Resource**
- **String Resources**
- **Style Resource**
- More Resource Types



String Resources

XML file saved at res/values/strings.xml:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

```
String string = getString(R.string.hello);
```

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

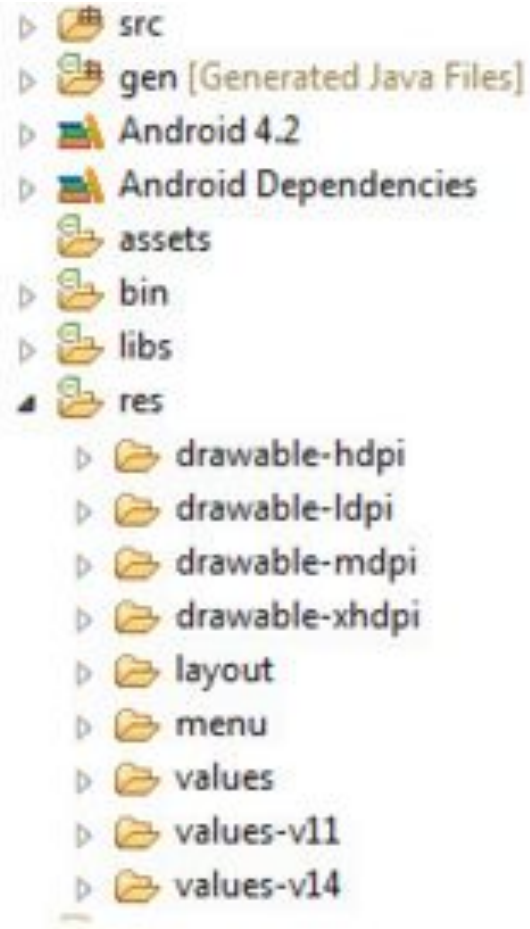
Style Resource

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomText" parent="@style/Text">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<EditText
    style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Resource Types

- Animation Resources
- Color State List Resource
- Drawable Resources
- Layout Resource
- Menu Resource
- String Resources
- Style Resource
- **More Resource Types**



More

- **Bool**: XML resource that carries a boolean value.
- **Color**: XML resource that carries a color value (a hexadecimal color).
- **Dimension**:XML resource that carries a dimension value (with a unit of measure).
- **ID**: XML resource that provides a unique identifier for application resources and components.
- **Integer**: XML resource that carries an integer value.
- **Integer Array**: XML resource that provides an array of integers.
- **Typed Array**: XML resource that provides a TypedArray (which you can use for an array of drawables).

Bool resource types

XML file saved at **res/values-small/bools.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="screen_small">true</bool>
    <bool name="adjust_view_bounds">true</bool>
</resources>
```

```
Resources res =  getResources() ;
boolean screenIsSmall = res. getBoolean (R.bool.screen_small);
```

```xml
<ImageView
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:src="@drawable/logo"
    android:adjustViewBounds="@bool/adjust_view_bounds" />
```

Color resource types

XML file saved at **res/values/colors.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <color name="translucent_red">#80ff0000</color>
</resources>
```

```
Resources res =  getResources() ;
int color = res. getColor (R.color.opaque_red);
```

```xml
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/translucent_red"
    android:text="Hello"/>
```

Dimension resource types

A dimension value defined in XML. A dimension is specified with a number followed by a unit of measure. For example: 10px, 2in, 5sp. The following units of measure are supported by Android

- dp
- sp
- pt
- px
- mm
- in

Dimension resource types

XML file saved at **res/values/dimens.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textview_height">25dp</dimen>
    <dimen name="textview_width">150dp</dimen>
    <dimen name="ball_radius">30dp</dimen>
    <dimen name="font_size">16sp</dimen>
</resources>
```

```
Resources res =  getResources() ;
float fontSize = res. getDimension (R.dimen.font_size);
```

```xml
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size"/>
```

ID resource types

XML file saved at **res/values/ids.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item type="id" name="button_ok" />
    <item type="id" name="dialog_exit" />
</resources>
```

```xml
<Button android:id="@id/button_ok"
    style="@style/button_style" />
```

Notice that the android:id value does not include the plus sign in the ID reference, because the ID already exists, as defined in the ids.xml example above. (When you specify an ID to an XML resource using the plus sign—in the format android:id="@+id/name"—it means that the "name" ID does not exist and should be created.)

Integer(or Integer array) resource types

XML file saved at **res/values/integers.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer name="max_speed">75</integer>
    <integer name="min_speed">5</integer>
</resources>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="bits">
        <item>4</item>
        <item>8</item>
        <item>16</item>
        <item>32</item>
    </integer-array>
</resources>
```

```
Resources res =  getResources() ;
int maxSpeed = res. getInteger (R.integer.max_speed);
```

```
Resources res =  getResources() ;
int[] bits = res. getIntArray (R.array.bits);
```

Typed array resource types

XML file saved at **res/values/arrays.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="icons">
        <item>@drawable/home</item>
        <item>@drawable/settings</item>
        <item>@drawable/logout</item>
    </array>
    <array name="colors">
        <item>#FFFF0000</item>
        <item>#FF00FF00</item>
        <item>#FF0000FF</item>
    </array>
</resources>
```

```
Resources res =  getResources() ;
TypedArray icons = res. obtainTypedArray (R.array.icons);
Drawable drawable = icons. getDrawable (0);

TypedArray colors = res. obtainTypedArray (R.array.colors);
int color = colors. getColor (0,0);
```