Android Working with XML Animations

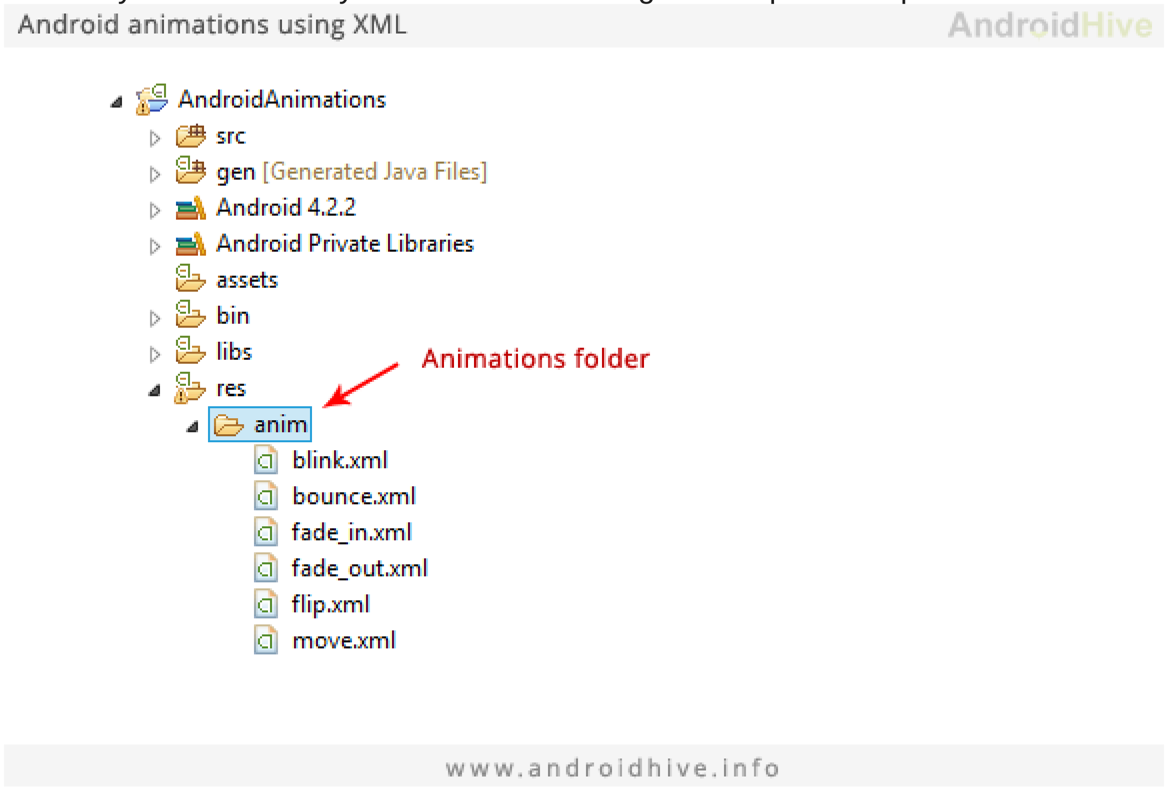
Adding animations to your app interface will give high quality feel to your android applications. Animations can be performed through either XML or android code. In this tutorial i explained how to do animations using XML notations.

Basic steps to perform animation

Following are the basic steps to perform an animation on any UI element. Creating animation is very simple, all it needs is creating necessary files and write small pieces of code.

Step 1: Create xml that defines the animation

Create an xml file which defines type of animation to perform. This file should be located under **anim** folder under **res** directory (**res ⇒ anim ⇒ animation.xml**). If you don't have anim folder in your res directory create one. Following is example of simple fade in animation.



fade_in.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0" />

</set>
```

Step 2: Load the animation

Next in your activity create an object of Animation class. And load the xml animation using AnimationUtils by calling **loadAnimation** function.

FadeInActivity.java

```
public class FadeInActivity extends Activity{
     TextView txtMessage;
     // Animation
    Animation animFadein;
     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fadein);

        txtMessage = (TextView) findViewById(R.id.txtMessage);

        // load the animation
        animFadein = AnimationUtils.loadAnimation(getApplicationContext(),
                R.anim.fade_in);
    }
}
```

Step 3: Set the animation listeners (Optional)

If you want to listen to animation events like start, end and repeat, your activity should implements AnimationListener. This step is optional. If you implement AnimationListener you will have to override following methods.

onAnimationStart – This will be triggered once the animation started
onAnimationEnd – This will be triggered once the animation is over
onAnimationRepeat – This will be triggered if the animation repeats

```
public class FadeInActivity extends Activity implements AnimationListener {
.
.
// set animation listener
animFadein.setAnimationListener(this);
.
.
// animation listeners
    @Override
    public void onAnimationEnd(Animation animation) {
        // Take any action after completing the animation
        // check for fade in animation
        if (animation == animFadein) {
            Toast.makeText(getApplicationContext(), "Animation Stopped",
                    Toast.LENGTH_SHORT).show();
        }


    }
```

```
@Override
    public void onAnimationRepeat(Animation animation) {
        // Animation is repeating
    }

    @Override
    public void onAnimationStart(Animation animation) {
        // Animation started
    }
```

Step 4: Finally start the animation

You can start animation whenever you want by calling **startAnimation** on any UI element by passing the type of animation. In this example i am calling fade in animation on TextView.

```
// start the animation
txtMessage.startAnimation(animFadein);
```

Complete Code

Following is complete code for FadeInActivity

FadeInActivity

```
package info.androidhive.androidanimations;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation.AnimationListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class FadeInActivity extends Activity implements AnimationListener {

    TextView txtMessage;
    Button btnStart;
     // Animation
    Animation animFadein;
     @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fadein);

        txtMessage = (TextView) findViewById(R.id.txtMessage);
        btnStart = (Button) findViewById(R.id.btnStart);
```

```
 // load the animation
        animFadein = AnimationUtils.loadAnimation(getApplicationContext(),
                R.anim.fade_in);

        // set animation listener
        animFadein.setAnimationListener(this);

        // button click event
        btnStart.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                txtMessage.setVisibility(View.VISIBLE);

                // start the animation
                txtMessage.startAnimation(animFadein);
            }
        });
    }

    @Override
    public void onAnimationEnd(Animation animation) {
        // Take any action after completing the animation

        // check for fade in animation
        if (animation == animFadein) {
            Toast.makeText(getApplicationContext(), "Animation Stopped",
                    Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onAnimationRepeat(Animation animation) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onAnimationStart(Animation animation) {
        // TODO Auto-generated method stub
    }

}
```

Important XML animation attributes

When working with animations it is better to have through knowledge about some of the important XML attributes which create major differentiation in animation behavior. Following are the important attributes you must known about.

android:duration – Duration of the animation in which the animation should complete

android:startOffset – It is the waiting time before an animation starts. This property is mainly used to perform multiple animations in a sequential manner

android:interpolator – It is the rate of change in animation

android:fillAfter – This defines whether to apply the animation transformation after the animation completes or not. If it sets to false the element changes to its previous state after the animation. This attribute should be use with **<set>** node

android:repeatMode – This is useful when you want your animation to be repeat

android:repeatCount – This defines number of repetitions on animation. If you set this value to **infinite** then animation will repeat infinite times

Some useful animations

Following i am giving xml code to perform lot of useful animations. Try to assign different values to xml attributes to see change in animations.

1. Fade In
2. Fade Out
3. Cross Fading
4. Blink
5. Zoom In
6. Zoom Out
7. Rotate
8. Move
9. Slide Up
10. Slide Down
11. Bounce
12. Sequential Animation
13. Together Animation

1. Fade In

For fade in animation you can use **<alpha>** tag which defines alpha value. Fade in animation is nothing but increasing alpha value from 0 to 1.

fade_in.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >
    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="1.0" />
</set>
```

2. Fade Out

Fade out is exactly opposite to fade in, where we need to decrease the alpha value from 1 to 0

fade_out.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <alpha
        android:duration="1000"
        android:fromAlpha="1.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:toAlpha="0.0" />

</set>
```

3. Cross Fading

Cross fading is performing fade in animation while other element is fading out. For this you don't have to create separate animation file, you can just use **fade_in.xml** and **fade_out.xml** files.

In the following code i loaded fade in and fade out, then performed them on two different UI elements.

```
TextView txtView1, txtView2;
Animation animFadeIn, animFadeOut;
.
.
// load animations
animFadeIn = AnimationUtils.loadAnimation(getApplicationContext(),
                R.anim.fade_in);
animFadeOut = AnimationUtils.loadAnimation(getApplicationContext(),
                R.anim.fade_out);
.
.
// set animation listeners
animFadeIn.setAnimationListener(this);
animFadeOut.setAnimationListener(this);
.
// Make fade in elements Visible first
txtMessage2.setVisibility(View.VISIBLE);
 // start fade in animation
txtMessage2.startAnimation(animFadeIn);
// start fade out animation
txtMessage1.startAnimation(animFadeOut);
```

4. Blink

Blink animation is animating fade out or fade in animation in repetitive fashion. For this you will have to set **android:repeatMode=”reverse”** and **android:repeatCount** attributes.

blink.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:duration="600"
        android:repeatMode="reverse"
        android:repeatCount="infinite"/>
</set>
```

5.Zoom In

For zoom use **<scale>** tag. Use **pivotX=”50%”** and **pivotY=”50%”** to perform zoom from the center of the element. Also you need to use **fromXScale**, **fromYScale** attributes which defines scaling of the object. Keep these value lesser than **toXScale**, **toYScale**

zoom_in.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >
    <scale
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="3"
        android:toYScale="3" >
    </scale>

</set>
```

6. Zoom Out

Zoom out animation is same as zoom in but **toXScale**, **toYScale** values are lesser than **fromXScale**, **fromYScale**

zoom_out.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >
    <scale
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="1000"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="0.5"
        android:toYScale="0.5" >
    </scale>

</set>
```

7. Rotate

Rotate animation uses **<rotate>** tag. For rotate animation required tags are **android:fromDegrees** and **android:toDegrees** which defines rotation angles.
Clock wise – use positive toDegrees value
Anti clock wise – use negative toDegrees value
rotate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="600"
        android:repeatMode="restart"
        android:repeatCount="infinite"
        android:interpolator="@android:anim/cycle_interpolator"/>

</set>
```

8. Move

In order to change position of object use **<translate>** tag. It uses **fromXDelta**, **fromYDelta** for X-direction and **toXDelta**, **toYDelta** attributes for Y-direction.
move.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">

   <translate
        android:fromXDelta="0%p"
        android:toXDelta="75%p"
        android:duration="800" />
</set>
```

9. Slide Up

Sliding animation uses **<scale>** tag only. Slide up can be achieved by setting **android:fromYScale="1.0"** and **android:toYScale="0.0"**

slide_up.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true" >

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="0.0" />

</set>
```

10. Slide Down

Slide down is exactly opposite to slide down animation. Just interchange **android:fromYScale** and **android:toYScale** values.

slide_down.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true">

    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="1.0" />

</set>
```

11. Bounce

Bounce is just an animation effect where animation ends in bouncing fashion. For this set **android:interpolator** value to **@android:anim/bounce_interpolator**. This bounce can be used with any kind animation. Following slide down example uses bounce effect.

bounce.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="0.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />

</set>
```

12. Sequential Animation

If you want to perform multiple animation in a sequential manner you have to use **android:startOffset** to give start delay time. The easy way to calculate this value is to **add the duration and startOffset** values of previous animation. Following is a sequential animation where set of move animations performs in sequential manner.

sequential.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >
    <!-- Use startOffset to give delay between animations -->
    <!-- Move -->
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="300"
        android:toXDelta="75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="1100"
        android:toYDelta="70%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="1900"
        android:toXDelta="-75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="2700"
        android:toYDelta="-70%p" />
```

```
<!-- Rotate 360 degrees -->
    <rotate
        android:duration="1000"
        android:fromDegrees="0"
        android:interpolator="@android:anim/cycle_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="3800"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:toDegrees="360" />

</set>
```

13. Together Animation

Performing all animation together is just writing all animations one by one without using **android:startOffset**

together.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >

    <scale
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="4000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="4"
        android:toYScale="4" >
    </scale>

    <!-- Rotate 180 degrees -->
    <rotate
        android:duration="500"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:toDegrees="360" />

</set>
```