View Animation

You can use the view animation system to perform tweened animation on Views. Tween animation calculates the animation with information such as the start point, end point, size, rotation, and other common aspects of an animation.

A tween animation can perform a series of simple transformations (position, size, rotation, and transparency) on the contents of a View object. So, if you have a `TextView` object, you can move, rotate, grow, or shrink the text. If it has a background image, the background image will be transformed along with the text. The `animation package` provides all the classes used in a tween animation.

A sequence of animation instructions defines the tween animation, defined by either XML or Android code. As with defining a layout, an XML file is recommended because it's more readable, reusable, and swappable than hard-coding the animation. In the example below, we use XML. (To learn more about defining an animation in your application code, instead of XML, refer to the `AnimationSet` class and other `Animation` subclasses.)

The animation instructions define the transformations that you want to occur, when they will occur, and how long they should take to apply. Transformations can be sequential or simultaneous - for example, you can have the contents of a TextView move from left to right, and then rotate 180 degrees, or you can have the text move and rotate simultaneously. Each transformation takes a set of parameters specific for that transformation (starting size and ending size for size change, starting angle and ending angle for rotation, and so on), and also a set of common parameters (for instance, start time and duration). To make several transformations happen simultaneously, give them the same start time; to make them sequential, calculate the start time plus the duration of the preceding transformation.

The animation XML file belongs in the `res/anim/` directory of your Android project. The file must have a single root element: this will be either a single `<alpha>`, `<scale>`, `<translate>`, `<rotate>`, interpolator element, or `<set>` element that holds groups of these elements (which may include another `<set>`). By default, all animation instructions are applied simultaneously. To make them occur sequentially, you must specify the `startOffset` attribute, as shown in the example below.

The following XML from one of the ApiDemos is used to stretch, then simultaneously spin and rotate a View object.

```
<set 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/decelerate_interpolator">
        <scale
           android:fromXScale="1.4"
           android:toXScale="0.0"
           android:fromYScale="0.6"
           android:toYScale="0.0"
           android:pivotX="50%"
           android:pivotY="50%"
           android:startOffset="700"
           android:duration="400"
           android:fillBefore="false" />
        <rotate
           android:fromDegrees="0"
           android:toDegrees="-45"
           android:toYScale="0.0"
           android:pivotX="50%"
           android:pivotY="50%"
           android:startOffset="700"
           android:duration="400" />
    </set>
</set>
```

Screen coordinates (not used in this example) are (0,0) at the upper left hand corner, and increase as you go down and to the right.

Some values, such as pivotX, can be specified relative to the object itself or relative to the parent. Be sure to use the proper format for what you want ("50" for 50% relative to the parent, or "50%" for 50% relative to itself).

You can determine how a transformation is applied over time by assigning an `Interpolator`. Android includes several Interpolator subclasses that specify various speed curves: for instance, `AccelerateInterpolator` tells a transformation to start slow and speed up. Each one has an attribute value that can be applied in the XML.

With this XML saved as `hyperspace_jump.xml` in the `res/anim/` directory of the project, the following code will reference it and apply it to an `ImageView` object from the layout.

```
ImageView spaceshipImage = (ImageView) findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation = AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

As an alternative to `startAnimation()`, you can define a starting time for the animation with `Animation.setStartTime()`, then assign the animation to the View with `View.setAnimation()`.

For more information on the XML syntax, available tags and attributes, see Animation Resources.

> **Note:** Regardless of how your animation may move or resize, the bounds of the View that holds your animation will not automatically adjust to accommodate it. Even so, the animation will still be drawn beyond the bounds of its View and will not be clipped. However, clipping *will occur* if the animation exceeds the bounds of the parent View.