Animation Resources

In this document

See also

- View Animation
- Property Animation

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/animator/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.animator.filename`
In XML: `@[package:]animator/filename`

SYNTAX:

```
<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>
```

The file must have a single root element: either `<set>`, `<objectAnimator>`, or `<valueAnimator>`. You can group animation elements together inside the `<set>` element, including other `<set>` elements.

ELEMENTS:

`<set>`

A container that holds other animation elements (`<objectAnimator>`, `<valueAnimator>`, or other `<set>` elements). Represents an `AnimatorSet`.

You can specify nested `<set>` tags to further group animations together. Each `<set>` can define its own `ordering` attribute.

attributes:

`android:ordering`

Keyword. Specifies the play ordering of animations in this set.

Value	Description
`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. Represents an `ObjectAnimator`.

attributes:

`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. The `objectAnimator` element does not expose a `target` attribute, however, so you cannot set the object to animate in the XML declaration. You have to inflate your animation XML resource by calling `loadAnimator()` and call `setTarget()` to set the target object that contains this property.

`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. If not specified, the animation starts at the value obtained by the property's get method. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:duration`

int. The time in milliseconds of the animation. 300 milliseconds is the default.

`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. How an animation behaves when it reaches the end of the animation. `android:repeatCount` must be set to a positive integer or `"-1"` for this attribute to have an effect. Set to `"reverse"` to have the animation reverse direction with each iteration or `"repeat"` to have the animation loop from the beginning each time.

`android:valueType`

Keyword. Do not specify this attribute if the value is a color. The animation framework automatically handles color values

Value	Description
`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. Represents a `ValueAnimator`.

attributes:

`android:valueTo`

float, int, or color. **Required**. The value where the animation ends. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:valueFrom`

float, int, or color. **Required**. The value where the animation starts. Colors are represented as six digit hexadecimal numbers (for example, #333333).

`android:duration`

int. The time in milliseconds of the animation. 300ms is the default.

`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. How an animation behaves when it reaches the end of the animation. `android:repeatCount` must be set to a positive integer or `"-1"` for this attribute to have an effect. Set to `"reverse"` to have the animation reverse direction with each iteration or `"repeat"` to have the animation loop from the beginning each time.

`android:valueType`

Keyword. Do not specify this attribute if the value is a color. The animation framework automatically handles color values.

Value	Description
`intType`	Specifies that the animated values are integers
`floatType` (default)	Specifies that the animated values are floats

EXAMPLE:

XML file saved at `res/animator/property_animator.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>
```

In order to run this animation, you must inflate the XML resources in your code to an `AnimatorSet` object, and then set the target objects for all of the animations before starting the animation set. Calling `setTarget()` sets a single target object for all children of the `AnimatorSet` as a convenience. The following code shows how to do this:

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

SEE ALSO:

- Property Animation
- API Demos for examples on how to use the property animation system.

View Animation

The view animation framework supports both tween and frame by frame animations, which can both be declared in XML. The following sections describe how to use both methods.

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:

```
<?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>
```

> 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).

ELEMENTS:

> `<set>`
>
> > A container that holds other animation elements (`<alpha>`, `<scale>`, `<translate>`, `<rotate>`) or other `<set>` elements. Represents an `AnimationSet`.

attributes:

`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. See the discussion below for more about Interpolators.

`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`.

attributes:

`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.

For more attributes supported by `<alpha>`, see the `Animation` class reference (of which, all XML attributes are inherrited by this element).

`<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`.

attributes:

`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.

For more attributes supported by `<scale>`, see the `Animation` class reference (of which, all XML attributes are inherrited by this

element).

`<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`.

attributes:

`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"`).

`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"`).

For more attributes supported by `<translate>`, see the `Animation` class reference (of which, all XML attributes are inherrited by this element).

`<rotate>`

A rotation animation. Represents a `RotateAnimation`.

attributes:

`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"`).

For more attributes supported by `<rotate>`, see the `Animation` class reference (of which, all XML attributes are inherrited by this element).

EXAMPLE:

XML file saved at `res/anim/hyperspace_jump.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>
</set>
```

This 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);
```

SEE ALSO:

- 2D Graphics: Tween Animation

Interpolators

An interpolator is an animation modifier defined in XML that affects the rate of change in an animation. This allows your existing animation effects to be accelerated, decelerated, repeated, bounced, etc.

An interpolator is applied to an animation element with the `android:interpolator` attribute, the value of which is a reference to an interpolator resource.

All interpolators available in Android are subclasses of the `Interpolator` class. For each interpolator class, Android includes a public resource you can reference in order to apply the interpolator to an animation using the `android:interpolator` attribute. The following table specifies the resource to use for each interpolator:

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`

Here's how you can apply one of these with the `android:interpolator` attribute:

```
<set android:interpolator="@android:anim/accelerate_interpolator">
    ...
</set>
```

Custom interpolators

If you're not satisfied with the interpolators provided by the platform (listed in the table above), you can create a custom interpolator resource with modified attributes. For example, you can adjust the rate of acceleration for the `AnticipateInterpolator`, or adjust the number of cycles for the `CycleInterpolator`. In order to do so, you need to create your own interpolator resource in an XML file.

FILE LOCATION:

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

COMPILED RESOURCE DATATYPE:

> Resource pointer to the corresponding interpolator object.

RESOURCE REFERENCE:

> In XML: `@[`*`package`*`:]anim/`*`filename`*

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<InterpolatorName xmlns:android="http://schemas.android.com/apk/res/android"
    android:attribute_name="value"
    />
```

> If you don't apply any attributes, then your interpolator will function exactly the same as those provided by the platform (listed in the table above).

ELEMENTS:

> Notice that each `Interpolator` implementation, when defined in XML, begins its name in lowercase.
>
> `<accelerateDecelerateInterpolator>`
>
> > The rate of change starts and ends slowly but accelerates through the middle.
> >
> > No attributes.
>
> `<accelerateInterpolator>`
>
> > The rate of change starts out slowly, then accelerates.
> >
> > attributes:
> >
> > `android:factor`

Float. The acceleration rate (default is 1).

`<anticipateInterpolator>`

The change starts backward then flings forward.

attributes:

`android:tension`

Float. The amount of tension to apply (default is 2).

`<anticipateOvershootInterpolator>`

The change starts backward, flings forward and overshoots the target value, then settles at the final value.

attributes:

`android:tension`

Float. The amount of tension to apply (default is 2).

`android:extraTension`

Float. The amount by which to multiply the tension (default is 1.5).

`<bounceInterpolator>`

The change bounces at the end.

No attributes

`<cycleInterpolator>`

Repeats the animation for a specified number of cycles. The rate of change follows a sinusoidal pattern.

attributes:

`android:cycles`

Integer. The number of cycles (default is 1).

`<decelerateInterpolator>`

The rate of change starts out quickly, then decelerates.

attributes:

`android:factor`

Float. The deceleration rate (default is 1).

`<linearInterpolator>`

The rate of change is constant.

No attributes.

`<overshootInterpolator>`

The change flings forward and overshoots the last value, then comes back.

attributes:

`android:tension`

Float. The amount of tension to apply (default is 2).

EXAMPLE:

XML file saved at `res/anim/my_overshoot_interpolator.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<overshootInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
    android:tension="7.0"
    />
```

This animation XML will apply the interpolator:

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@anim/my_overshoot_interpolator"
    android:fromXScale="1.0"
    android:toXScale="3.0"
    android:fromYScale="1.0"
    android:toYScale="3.0"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="700" />
```

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>
```

ELEMENTS:

`<animation-list>`

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

attributes:

`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.

attributes:

`android:drawable`

Drawable resource. The drawable to use for this frame.

`android:duration`

Integer. The duration to show this frame, in milliseconds.

EXAMPLE:

XML file saved at `res/anim/rocket.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>
```

This application code will set the animation as the background for a View, then play the animation:

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

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

SEE ALSO:

- 2D Graphics: Frame Animation