Drawable Resources

See also
2D Graphics

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`.

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`.

Inset Drawable
: An XML file that defines a drawable that insets another drawable by a specified distance. This is useful when a View needs a background drawble that is smaller than the View's actual bounds.

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`.

Also see the Animation Resource document for how to create an `AnimationDrawable`.

> **Note:** A color resource can also be used as a drawable in XML. For example, when creating a state list drawable, you can reference a color resource for the `android:drawable` attribute (`android:drawable="@color/green"`).

Bitmap

A bitmap image. Android supports bitmap files in a three formats: `.png` (preferred), `.jpg` (acceptable), `.gif`

(discouraged).

You can reference a bitmap file directly, using the filename as the resource ID, or create an alias resource ID in XML.

> **Note:** Bitmap files may be automatically optimized with lossless image compression by the `aapt` tool during the build process. For example, a true-color PNG that does not require more than 256 colors may be converted to an 8-bit PNG with a color palette. This will result in an image of equal quality but which requires less memory. So be aware that the image binaries placed in this directory can change during the build. If you plan on reading an image as a bit stream in order to convert it to a bitmap, put your images in the `res/raw/` folder instead, where they will not be optimized.

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`

EXAMPLE:

With an image saved at `res/drawable/myimage.png`, this layout XML applies the image to a View:

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

The following application code retrieves the image as a `Drawable`:

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

SEE ALSO:

- 2D Graphics
- `BitmapDrawable`

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.

> **Note:** You can use a `<bitmap>` element as a child of an `<item>` element. For example, when creating a state list or layer list, you can exclude the `android:drawable` attribute from an `<item>` element and nest a `<bitmap>` inside it that defines the drawable item.

FILE LOCATION:

`res/drawable/filename.xml`
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`

SYNTAX:

```
<?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:tileMode=["disabled" | "clamp" | "repeat" | "mirror"] />
```

ELEMENTS:

`<bitmap>`

Defines the bitmap source and its properties.

ATTRIBUTES:

`xmlns:android`

String. Defines the XML namespace, which must be `"http://schemas.android.com/apk/res/android"`. This is required only if the `<bitmap>` is the root element—it is not needed when the `<bitmap>` is nested inside an `<item>`.

`android:src`

Drawable resource. **Required**. Reference to a drawable resource.

`android:antialias`

Boolean. Enables or disables antialiasing.

`android:dither`

Boolean. Enables or disables dithering of the bitmap if the bitmap does not have the same pixel configuration as the screen (for instance: a ARGB 8888 bitmap with an RGB 565 screen).

`android:filter`

Boolean. Enables or disables bitmap filtering. Filtering is used when the bitmap is shrunk or stretched to smooth its apperance.

`android:gravity`

Keyword. Defines the gravity for the bitmap. The gravity indicates where to position the drawable in its container if the bitmap is smaller than the container.

Must be one or more (separated by '|') of the following constant values:

Value	Description
`top`	Put the object at the top of its container, not changing its size.
`bottom`	Put the object at the bottom of its container, not changing its size.
`left`	Put the object at the left edge of its container, not changing its size.
`right`	Put the object at the right edge of its container, not changing its size.
`center_vertical`	Place object in the vertical center of its container, not changing its size.

`fill_vertical`	Grow the vertical size of the object if needed so it completely fills its container.
`center_horizontal`	Place object in the horizontal center of its container, not changing its size.
`fill_horizontal`	Grow the horizontal size of the object if needed so it completely fills its container.
`center`	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
`fill`	Grow the horizontal and vertical size of the object if needed so it completely fills its container. This is the default.
`clip_vertical`	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip is based on the vertical gravity: a top gravity clips the bottom edge, a bottom gravity clips the top edge, and neither clips both edges.
`clip_horizontal`	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip is based on the horizontal gravity: a left gravity clips the right edge, a right gravity clips the left edge, and neither clips both edges.

`android:tileMode`

Keyword. Defines the tile mode. When the tile mode is enabled, the bitmap is repeated. Gravity is ignored when the tile mode is enabled.

Must be one of the following constant values:

Value	Description
`disabled`	Do not tile the bitmap. This is the default value.
`clamp`	Replicates the edge color if the shader draws outside of its original bounds
`repeat`	Repeats the shader's image horizontally and vertically.
`mirror`	Repeats the shader's image horizontally and vertically, alternating mirror images so that adjacent images always seam.

EXAMPLE:

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/icon"
    android:tileMode="repeat" />
```

SEE ALSO:

- `BitmapDrawable`
- Creating alias resources

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.

Same as with a normal bitmap, you can reference a Nine-Patch file directly or from a resource defined by XML.

For a complete discussion about how to create a Nine-Patch file with stretchable regions, see the 2D Graphics document.

Nine-Patch File

FILE LOCATION:

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

COMPILED RESOURCE DATA TYPE:

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`, this layout XML applies the Nine-Patch to a View:

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

SEE ALSO:

- 2D Graphics
- `NinePatchDrawable`

XML Nine-Patch

An XML Nine-Patch is a resource defined in XML that points to a Nine-Patch file. The XML can specify dithering for the image.

FILE LOCATION:

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

COMPILED RESOURCE DATA TYPE:

Resource pointer to a `NinePatchDrawable`.

RESOURCE REFERENCE:

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

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<nine-patch
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:dither=["true" | "false"] />
```

ELEMENTS:

`<nine-patch>`
Defines the Nine-Patch source and its properties.

ATTRIBUTES:

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

`android:src`
Drawable resource. **Required**. Reference to a Nine-Patch file.

`android:dither`
Boolean. Enables or disables dithering of the bitmap if the bitmap does not have the same pixel configuration as the screen (for instance: a ARGB 8888 bitmap with an RGB 565 screen).

EXAMPLE:

```
<?xml version="1.0" encoding="utf-8"?>
<nine-patch xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/myninepatch"
    android:dither="false" />
```

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.

Each drawable is represented by an `<item>` element inside a single `<layer-list>` element.

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>
```

ELEMENTS:

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

ATTRIBUTES:

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

`<item>`

Defines a drawable to place in the layer drawable, in a position defined by its attributes. Must be a child of a `<selector>` element. Accepts child `<bitmap>` elements.

ATTRIBUTES:

`android:drawable`
Drawable resource. **Required**. Reference to a drawable resource.

`android:id`
Resource ID. A unique resource ID for this drawable. To create a new resource ID for this item, use the form: `"@+id/name"`. The plus symbol indicates that this should be created as a new ID. You can use this identifier to retrieve and modify the drawable with `View.findViewById()` or `Activity.findViewById()`.

`android:top`
Integer. The top offset in pixels.

`android:right`
Integer. The right offset in pixels.

`android:bottom`
Integer. The bottom offset in pixels.

`android:left`
Integer. The left offset in pixels.

All drawable items are scaled to fit the size of the containing View, by default. Thus, placing your images in a layer list at different positions might increase the size of the View and some images scale as appropriate. To avoid scaling items in the list, use a `<bitmap>` element inside the `<item>` element to specify the drawable and define the gravity to something that does not scale, such as `"center"`. For example, the following `<item>` defines an item that scales to fit its container View:

```
<item android:drawable="@drawable/image" />
```

To avoid scaling, the following example uses a `<bitmap>` element with centered gravity:

```
<item>
  <bitmap android:src="@drawable/image"
          android:gravity="center" />
</item>
```

EXAMPLE:

XML file saved at `res/drawable/layers.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>
```

Notice that this example uses a nested `<bitmap>` element to define the drawable resource for each item with a "center" gravity. This ensures that none of the images are scaled to fit the size of the container, due to resizing caused by the offset images.

This layout XML applies the drawable to a View:

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

The result is a stack of increasingly offset images:

SEE ALSO:

- `LayerDrawable`

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 niether) and, using a state list drawable, you can provide a different background image for each state.

You can describe the state list in an XML file. Each graphic is represented by an `<item>` element inside a single `<selector>` element. Each `<item>` uses various attributes to describe the state in which it should be used as the graphic for the drawable.

During each state change, the state list is traversed top to bottom and the first item that matches the current state is used—the selection is *not* based on the "best match," but simply the first item that meets the minimum criteria of the state.

FILE LOCATION:

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

COMPILED RESOURCE DATA TYPE:

Resource pointer to a `StateListDrawable`.

RESOURCE REFERENCE:

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

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_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>
```

ELEMENTS:

`<selector>`

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

ATTRIBUTES:

`xmlns:android`

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

`android:constantSize`

Boolean. "true" if the drawable's reported internal size remains constant as the state changes (the size is the maximum of all of the states); "false" if the size varies based on the current state. Default is false.

`android:dither`

Boolean. "true" to enable dithering of the bitmap if the bitmap does not have the same pixel configuration as the screen (for instance, an ARGB 8888 bitmap with an RGB 565 screen); "false" to disable dithering. Default is true.

`android:variablePadding`

Boolean. "true" if the drawable's padding should change based on the current state that is selected; "false" if the padding should stay the same (based on the maximum padding of all the states). Enabling this feature requires that you deal with performing layout when the state changes, which is often not supported. Default is false.

`<item>`

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

ATTRIBUTES:

`android:drawable`

Drawable resource. **Required**. Reference to a drawable resource.

`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.

`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.)

`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.

`android:state_window_focused`
Boolean. "true" if this item should be used when the application window has focus (the application is in the foreground), "false" if this item should be used when the application window does not have focus (for example, if the notification shade is pulled down or a dialog appears).

> **Note:** Remember that Android applies the first item in the state list that matches the current state of the object. So, if the first item in the list contains none of the state attributes above, then it is applied every time, which is why your default value should always be last (as demonstrated in the following example).

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:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

This layout XML applies the state list drawable to a Button:

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

SEE ALSO:

- StateListDrawable

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.

FILE LOCATION:

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

COMPILED RESOURCE DATATYPE:

Resource pointer to a `LevelListDrawable`.

RESOURCE REFERENCE:

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

SYNTAX:

```
<?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>
```

ELEMENTS:

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

ATTRIBUTES:

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

`<item>`
Defines a drawable to use at a certain level.

ATTRIBUTES:

`android:drawable`
Drawable resource. **Required**. Reference to a drawable resource to be inset.

`android:maxLevel`
Integer. The maximum level allowed for this item.

`android:minLevel`
Integer. The minimum level allowed for this item.

EXAMPLE:

```
<?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>
```

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

SEE ALSO:

- `LevelListDrawable`

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()`.

FILE LOCATION:

`res/drawable/filename.xml`

The filename is used as the resource ID.

COMPILED RESOURCE DATA TYPE:
Resource pointer to a `TransitionDrawable`.

RESOURCE REFERENCE:
In Java: `R.drawable.filename`
In XML: `@[package:]drawable/filename`

SYNTAX:

```
<?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>
```

ELEMENTS:

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

ATTRIBUTES:

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

`<item>`
Defines a drawable to use as part of the drawable transition. Must be a child of a `<transition>` element. Accepts child `<bitmap>` elements.

ATTRIBUTES:

`android:drawable`
Drawable resource. **Required**. Reference to a drawable resource.

`android:id`
Resource ID. A unique resource ID for this drawable. To create a new resource ID for this item, use the form: `"@+id/name"`. The plus symbol indicates that this should be created as a new ID. You can use this identifier to retrieve and modify the drawable with `View.findViewById()` or `Activity.findViewById()`.

`android:top`
Integer. The top offset in pixels.

`android:right`
Integer. The right offset in pixels.

`android:bottom`
Integer. The bottom offset in pixels.

`android:left`
Integer. The left offset in pixels.

EXAMPLE:
XML file saved at `res/drawable/transition.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>
```

This layout XML applies the drawable to a View:

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

And the following code performs a 500ms transition from the first item to the second:

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

SEE ALSO:

- TransitionDrawable

Inset Drawable

A drawable defined in XML that insets another drawable by a specified distance. This is useful when a View needs a background that is smaller than the View's actual bounds.

FILE LOCATION:

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

COMPILED RESOURCE DATATYPE:

Resource pointer to a InsetDrawable.

RESOURCE REFERENCE:

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

SYNTAX:

```
<?xml version="1.0" encoding="utf-8"?>
<inset
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/drawable_resource"
    android:insetTop="dimension"
    android:insetRight="dimension"
    android:insetBottom="dimension"
    android:insetLeft="dimension" />
```

ELEMENTS:

`<inset>`
Defines the inset drawable. This must be the root element.

ATTRIBUTES:

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

`android:drawable`
Drawable resource. **Required**. Reference to a drawable resource to be inset.

`android:insetTop`
Dimension. The top inset, as a dimension value or dimension resource

`android:insetRight`
Dimension. The right inset, as a dimension value or dimension resource

`android:insetBottom`
Dimension. The bottom inset, as a dimension value or dimension resource

`android:insetLeft`
Dimension. The left inset, as a dimension value or dimension resource

EXAMPLE:

```
<?xml version="1.0" encoding="utf-8"?>
<inset xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/background"
    android:insetTop="10dp"
    android:insetLeft="10dp" />
```

SEE ALSO:

- `InsetDrawable`

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.

FILE LOCATION:
`res/drawable/filename.xml`
The filename is used as the resource ID.

COMPILED RESOURCE DATATYPE:
Resource pointer to a `ClipDrawable`.

RESOURCE REFERENCE:
In Java: `R.drawable.filename`
In XML: `@[package:]drawable/filename`

SYNTAX:

```
<?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"]
/>
```

ELEMENTS:

`<clip>`

Defines the clip drawable. This must be the root element.

ATTRIBUTES:

`xmlns:android`

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

`android:drawable`

Drawable resource. **Required**. Reference to a drawable resource to be clipped.

`android:clipOrientation`

Keyword. The orientation for the clip.

Must be one of the following constant values:

Value	Description
`horizontal`	Clip the drawable horizontally.
`vertical`	Clip the drawable vertically.

`android:gravity`

Keyword. Specifies where to clip within the drawable.

Must be one or more (separated by '|') of the following constant values:

Value	Description
`top`	Put the object at the top of its container, not changing its size. When `clipOrientation` is `"vertical"`, clipping occurs at the bottom of the drawable.
`bottom`	Put the object at the bottom of its container, not changing its size. When `clipOrientation` is `"vertical"`, clipping occurs at the top of the drawable.
`left`	Put the object at the left edge of its container, not changing its size. This is the default. When `clipOrientation` is `"horizontal"`, clipping occurs at the right side of the drawable. This is the default.
`right`	Put the object at the right edge of its container, not changing its size. When `clipOrientation` is `"horizontal"`, clipping occurs at the left side of the drawable.
`center_vertical`	Place object in the vertical center of its container, not changing its size. Clipping behaves the same as when gravity is `"center"`.
`fill_vertical`	Grow the vertical size of the object if needed so it completely fills its container. When `clipOrientation` is `"vertical"`, no clipping occurs because the drawable fills the vertical space (unless the drawable level is 0, in which case it's not visible).
`center_horizontal`	Place object in the horizontal center of its container, not changing its size. Clipping behaves the same as when gravity is `"center"`.
`fill_horizontal`	Grow the horizontal size of the object if needed so it completely fills its container. When `clipOrientation` is `"horizontal"`, no clipping occurs because the drawable fills the horizontal space (unless the drawable level is 0, in which case it's not visible).

`center`	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size. When `clipOrientation` is `"horizontal"`, clipping occurs on the left and right. When `clipOrientation` is `"vertical"`, clipping occurs on the top and bottom.
`fill`	Grow the horizontal and vertical size of the object if needed so it completely fills its container. No clipping occurs because the drawable fills the horizontal and vertical space (unless the drawable level is 0, in which case it's not visible).
`clip_vertical`	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip is based on the vertical gravity: a top gravity clips the bottom edge, a bottom gravity clips the top edge, and neither clips both edges.
`clip_horizontal`	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip is based on the horizontal gravity: a left gravity clips the right edge, a right gravity clips the left edge, and neither clips both edges.

EXAMPLE:

XML file saved at `res/drawable/clip.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" />
</clip>
```

The following layout XML applies the clip drawable to a View:

```
<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);
```

Increasing the level reduces the amount of clipping and slowly reveals the image. Here it is at a level of 7000:

> **Note:** The default level is 0, which is fully clipped so the image is not visible. When the level is 10,000, the image is not clipped and completely visible.

SEE ALSO:

- `ClipDrawable`

Scale Drawable

A drawable defined in XML that changes the size of another drawable based on its current level.

FILE LOCATION:

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

COMPILED RESOURCE DATATYPE:

Resource pointer to a `ScaleDrawable`.

RESOURCE REFERENCE:

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

SYNTAX:

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

ELEMENTS:

`<scale>`
Defines the scale drawable. This must be the root element.

ATTRIBUTES:

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

`android:drawable`
Drawable resource. **Required**. Reference to a drawable resource.

`android:scaleGravity`
Keyword. Specifies the gravity position after scaling.

Must be one or more (separated by '|') of the following constant values:

Value	Description
`top`	Put the object at the top of its container, not changing its size.
`bottom`	Put the object at the bottom of its container, not changing its size.
`left`	Put the object at the left edge of its container, not changing its size. This is the default.
`right`	Put the object at the right edge of its container, not changing its size.
`center_vertical`	Place object in the vertical center of its container, not changing its size.

`fill_vertical`	Grow the vertical size of the object if needed so it completely fills its container.
`center_horizontal`	Place object in the horizontal center of its container, not changing its size.
`fill_horizontal`	Grow the horizontal size of the object if needed so it completely fills its container.
`center`	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
`fill`	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
`clip_vertical`	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip is based on the vertical gravity: a top gravity clips the bottom edge, a bottom gravity clips the top edge, and neither clips both edges.
`clip_horizontal`	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip is based on the horizontal gravity: a left gravity clips the right edge, a right gravity clips the left edge, and neither clips both edges.

`android:scaleHeight`
Percentage. The scale height, expressed as a percentage of the drawable's bound. The value's format is XX%. For instance: 100%, 12.5%, etc.

`android:scaleWidth`
Percentage. The scale width, expressed as a percentage of the drawable's bound. The value's format is XX%. For instance: 100%, 12.5%, etc.

EXAMPLE:

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/logo"
    android:scaleGravity="center_vertical|center_horizontal"
    android:scaleHeight="80%"
    android:scaleWidth="80%" />
```

SEE ALSO:

- ScaleDrawable

Shape Drawable

This is a generic shape defined in XML.

FILE LOCATION:
`res/drawable/filename.xml`
The filename is used as the resource ID.

COMPILED RESOURCE DATATYPE:
Resource pointer to a ShapeDrawable.

RESOURCE REFERENCE:
In Java: `R.drawable.filename`
In XML: `@[package:]drawable/filename`

SYNTAX:

```
<?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:usesLevel=["true" | "false"] />
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <size
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
    <solid
        android:color="color" />
    <stroke
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
</shape>
```

ELEMENTS:

`<shape>`

The shape drawable. This must be the root element.

ATTRIBUTES:

`xmlns:android`

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

`android:shape`

Keyword. Defines the type of shape. Valid values are:

Value	Desciption
`"rectangle"`	A rectangle that fills the containing View. This is the default shape.
`"oval"`	An oval shape that fits the dimensions of the containing View.
`"line"`	A horizontal line that spans the width of the containing View. This shape

	requires the `<stroke>` element to define the width of the line.
`"ring"`	A ring shape.

The following attributes are used only when `android:shape="ring"`:

`android:innerRadius`
Dimension. The radius for the inner part of the ring (the hole in the middle), as a dimension value or dimension resource.

`android:innerRadiusRatio`
Float. The radius for the inner part of the ring, expressed as a ratio of the ring's width. For instance, if `android:innerRadiusRatio="5"`, then the inner radius equals the ring's width divided by 5. This value is overridden by `android:innerRadius`. Default value is 9.

`android:thickness`
Dimension. The thickness of the ring, as a dimension value or dimension resource.

`android:thicknessRatio`
Float. The thickness of the ring, expressed as a ratio of the ring's width. For instance, if `android:thicknessRatio="2"`, then the thickness equals the ring's width divided by 2. This value is overridden by `android:innerRadius`. Default value is 3.

`android:useLevel`
Boolean. "true" if this is used as a `LevelListDrawable`. This should normally be "false" or your shape may not appear.

`<corners>`
Creates rounded corners for the shape. Applies only when the shape is a rectangle.

ATTRIBUTES:

`android:radius`
Dimension. The radius for all corners, as a dimension value or dimension resource. This is overridden for each corner by the following attributes.

`android:topLeftRadius`
Dimension. The radius for the top-left corner, as a dimension value or dimension resource.

`android:topRightRadius`
Dimension. The radius for the top-right corner, as a dimension value or dimension resource.

`android:bottomLeftRadius`
Dimension. The radius for the bottom-left corner, as a dimension value or dimension resource.

`android:bottomRightRadius`
Dimension. The radius for the bottom-right corner, as a dimension value or dimension resource.

> **Note:** Every corner must (initially) be provided a corner radius greater than 1, or else no corners are rounded. If you want specific corners to *not* be rounded, a work-around is to use `android:radius` to set a default corner radius greater than 1, but then override each and every corner with the values you really want, providing zero ("0dp") where you don't want rounded corners.

`<gradient>`
Specifies a gradient color for the shape.

ATTRIBUTES:

`android:angle`
Integer. The angle for the gradient, in degrees. 0 is left to right, 90 is bottom to top. It must be a multiple of 45. Default is 0.

`android:centerX`
Float. The relative X-position for the center of the gradient (0 - 1.0). Does not apply when `android:type="linear"`.

`android:centerY`
Float. The relative Y-position for the center of the gradient (0 - 1.0). Does not apply when

android:type="linear".

android:centerColor
Color. Optional color that comes between the start and end colors, as a hexadecimal value or color resource.

android:endColor
Color. The ending color, as a hexadecimal value or color resource.

android:gradientRadius
Float. The radius for the gradient. Only applied when android:type="radial".

android:startColor
Color. The starting color, as a hexadecimal value or color resource.

android:type
Keyword. The type of gradient pattern to apply. Valid values are:

Value	Description
"linear"	A linear gradient. This is the default.
"radial"	A radial gradient. The start color is the center color.
"sweep"	A sweeping line gradient.

android:useLevel
Boolean. "true" if this is used as a LevelListDrawable.

<padding>
Padding to apply to the containing View element (this pads the position of the View content, not the shape).

ATTRIBUTES:

android:left
Dimension. Left padding, as a dimension value or dimension resource.

android:top
Dimension. Top padding, as a dimension value or dimension resource.

android:right
Dimension. Right padding, as a dimension value or dimension resource.

android:bottom
Dimension. Bottom padding, as a dimension value or dimension resource.

<size>
The size of the shape.

ATTRIBUTES:

android:height
Dimension. The height of the shape, as a dimension value or dimension resource.

android:width
Dimension. The width of the shape, as a dimension value or dimension resource.

> **Note:** The shape scales to the size of the container View proportionate to the dimensions defined here, by default. When you use the shape in an ImageView, you can restrict scaling by setting the android:scaleType to "center".

<solid>
A solid color to fill the shape.

ATTRIBUTES:

android:color
Color. The color to apply to the shape, as a hexadecimal value or color resource.

`<stroke>`
A stroke line for the shape.

ATTRIBUTES:

`android:width`
Dimension. The thickness of the line, as a dimension value or dimension resource.

`android:color`
Color. The color of the line, as a hexadecimal value or color resource.

`android:dashGap`
Dimension. The distance between line dashes, as a dimension value or dimension resource. Only valid if `android:dashWidth` is set.

`android:dashWidth`
Dimension. The size of each dash line, as a dimension value or dimension resource. Only valid if `android:dashGap` is set.

EXAMPLE:

XML file saved at `res/drawable/gradient_box.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>
```

This layout XML applies the shape drawable to a View:

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

This application code gets the shape drawable and applies it to a View:

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

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

SEE ALSO:

- ShapeDrawable

← Back to Resource Types ↑ Go to top