Drawable. Изображения, фигуры и градиенты

Ресурсы изображений

Android генерирует идентификаторы ресурсов для файлов изображений, расположенных в подкаталоге **/res/drawable**. Поддерживаются файлы PNG (самый предпочтительный), GIF, JPG. Для каждого файла изображения, который находится в этом каталоге, генерируется уникальный идентификатор на основе имени файла без расширения. Например, если у файла имя "cat.jpg", то для него будет создан идентификатор ресурса **R.drawable.cat**. Нужно следить за уникальностью имён, так как если у вас будут два файла "cat.jpg" и "cat.png", то возникнет ошибка. Кроме того, не нужно создавать дополнительных подкаталогов в папке **res/drawable**, так как файлы оттуда не будут считываться.

Можно адаптировать картинки для разных разрешений и размеров экранов. Для этого нужно создать специальные папки и складывать там нужные картинки. Например, для различных разрешений экрана используются папки **drawable-hdpi**, **drawable-mdpi**, **drawable-ldpi** и пр.. Существует аналогичное деление для размеров экрана: **drawable-normal**, **drawable-large** и т.д.. Можно совмещать эти способы и создавать папки типа **drawable-normal-hdpi**. Для изображений, которые должны оставаться неизменными вне зависимости от разрешения экрана, следует создать папку **drawable-nodpi**.

Для памятки приведу используемые размеры изображений для значков и фона экрана:

* res/drawable-ldpi - (120 DPI) (QVGA):  
  ic\_launcher.png (значок), 36 × 36 background.png (фон), 320 × 240 pixels
* res/drawable-mdpi - (160 DPI) (HVGA):  
  background.png, 320 × 480
* res/drawable-hdpi - (240 DPI) (WVGA):  
  ic\_launcher.png, 72 × 72  
  background.png, 800 × 480
* res/drawable-xhdpi (320 DPI) (WSVGA or HDTV):  
  ic\_launcher.png, 96 × 96  
  background.png, 1024 × 600 или 1280 × 720 pixels

Чтобы сослаться на изображение из **res/drawable** в XML-файлах разметки, используйте следующий синтаксис:

<Button

android:id="@+id/button1"

...

android:background="@drawable/cat"

</Button>

Обратите внимание, что в любом случае мы обращаемся к ресурсу как **drawable**, а не **drawable-hdpi** или как-то еще.

Программным способом можно достучаться до изображения следующим образом:

// вызываем getDrawable для получения изображения

BitmapDrawable bd = activity.getResources().getDrawable(R.drawable.cat);

// Затем можно использовать полученный объект, чтобы установить фон

button.setBackgroundDrawable(bd);

// или можно установить фон непосредственно по идентификатору ресурса

button.setBackgroundResource(R.drawable.icon);

Android включает простые ресурсы для рисования, которые можно полностью описать в формате XML. Это касается классов **ColorDrawable**, **ShapeDrawable** и **GradientDrawable**. Данные ресурсы хранятся в каталоге **res/drawable** и могут быть идентифицированы в коде приложения по именам файлов, записанным в нижнем регистре.

Если описывать эти ресурсы в формате XML и указывать атрибуты для них с помощью аппаратно-независимых пикселей (density-independent pixels), система сможет их плавно масштабировать. Как и в случае с векторной графикой, эти ресурсы могут динамически масштабироваться, отображаясь корректно и без артефактов при любых размерах и разрешениях экрана, независимо от плотности пикселов. Исключение — ресурс **GradientDrawable**, радиус для которого должен быть указан в пикселях.

Получить имя ресурса

Иногда нужно получить не сам идентификатор, а его имя в виде **R.drawable.cat**, что сохранить его, скажем, в базе данных. Воспользуйтесь следующим приёмом:

getResources().getIdentifier("image\_name","drawable", getPackageName())

getResources().getIdentifier("your.full.package.name:drawable/image\_name", null, null);

Бывает и обратная задача - из имени ресурса получить идентификатор. Тот же принцип.

String mDrawableName = "cat1"; // файл cat1.png в папке drawable

int resID = getResources().getIdentifier(mDrawableName , "drawable", getPackageName());

Строго говоря это относится не только к типу **drawable**, но обычно приходится встречаться с проблемой именно для них.

<animated-rotate> (класс AnimatedRotateDrawable)

Ресурсы для создания эффекта вращения.

|  |  |  |  |
| --- | --- | --- | --- |
| **Атрибут** | **Тип** | **Значение по умолчанию** | **Описание** |
| visible | boolean | parent|true | Determines if drawable is visible. |
| frameDuration | integer | 150 | The duration of each frame, in milliseconds. |
| framesCount | integer | 12 | Number of frames of rotation to animate. |
| pivotX | float|fraction | .5 | The pivot point, as a fraction of the width. |
| pivotY | float|fraction | .5 | The pivot point, as a fraction of the height. |
| drawable | reference | null | The drawable to use for this item. Either this must be present or a drawable subelement must exist. |

Пример использования в [индикаторе прогресса](http://developer.alexanderklimov.ru/android/views/progressbar.php#stylebar).

<animation-list> (класс AnimationDrawable)

Вращение по кадрам. Нужно подготовить несколько похожих изображений, которые будут сменять друг друга.

|  |  |  |  |
| --- | --- | --- | --- |
| **Атрибут** | **Тип** | **Значение по умолчанию** | **Описание** |
| visible | boolean | parent|true | Determines if drawable is visible. |
| variablePadding | boolean | false | If true, allows the drawable's padding to change based on the current state that is selected. |
| oneshot | boolean | false | If true, the animation will only run a single time and then stop. |

Если у нас имеется несколько изображений одного ProgressBar



<?xml version="1.0" encoding="utf-8"?>

<animation-list android:oneshot="false"

xmlns:android="schemas.android.com/apk/res/android">

<item android:duration="100">

<scale android:drawable="@drawable/blue\_1" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_2" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_3" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_4" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_5" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_6" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_7" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_8" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_9" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_10" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_11" android:scaleGravity="center" />

</item>

<item android:duration="100">

<scale android:drawable="@drawable/blue\_12" android:scaleGravity="center" />

</item>

</animation-list>

[Отсюда](http://habrahabr.ru/post/147090/)

<bitmap> (класс BitmapDrawable)

Позволяет использовать растровые изображения и проделывать с ними различные операции: растягивать, размножать, выравнивать. Часто используется в составе других ресурсов. Смотри примеры ниже.

Также смотри пример [Как заполнить фон повторяющимся изображением (черепицей)](http://developer.alexanderklimov.ru/android/design/tilemode.php#tilemode)

Основной атрибут **src**, в котором указывается файл изображения.

Атрибут **gravity** позволяет управлять размещением картинки внутри контейнера. Можно использовать несколько значений, разделенных знаком **|**: **top**, **left**, **center**, **fill** и др.

<?xml version="1.0" encoding="utf-8"?>

<bitmap

xmlns:android="http://schemas.android.com/apk/res/android"

android:gravity="top | left"

android:src="@drawable/ic\_launcher">

</bitmap>

У него также есть интересный атрибут **tileMode**, позволяющий замостить изображением всё доступное пространство. Можно использовать значения **repeat**, **mirror**, **clamp**. Пример есть выше по ссылке.

Другие атрибуты: **antialias** (сглаживание), **dither** (преобразование цветов, если текущей палитры недостаточно для отображения), **filter** (фильтр при сжатии или растягивании), **mipMap**.

В Android 5.0 (API 21) у **bitmap** появился новый атрибут **android:tint**, позволяющий [задавать оттенки](http://developer.alexanderklimov.ru/android/tint.php).

<clip> (класс ClipDrawable)

Синтаксис

<?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"] />

Данный вид ресурсов часто используется для создания [индикаторов прогресса](http://developer.alexanderklimov.ru/android/views/progressbar.php#vertical).

Тег **clip** позволяет обрезать **Drawable** по горизонтальной или по вертикальной оси через атрибут **clipOrientation**.

Атрибут **gravity** позволяет указать направление обрезания. Значения атрибута можно комбинировать. Например, если у **clipOrientation** значение равно **vertical**, а у **gravity** - **top**, то отрежется нижняя часть изображения.

Создадим файл **res/drawable/clipping.xml**

<?xml version="1.0" encoding="utf-8"?>

<clip xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/cosmos"

android:clipOrientation="horizontal"

android:gravity="left" />

Присвоим созданный ресурс компоненту **ImageView**:

<ImageView

android:id="@+id/imageViewClip"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_gravity="center\_horizontal"

android:src="@drawable/clipping" />

Сама обрезка производится программно через метод **setLevel()** с диапазоном значений от 0 до 10000, где 0 - картинка полностью обрезана и не видна, 10000 - картинка видна полностью.

public void onClick(View view) {

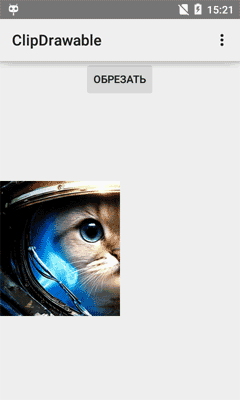
ImageView imageView = (ImageView) findViewById(R.id.imageViewClip);

ClipDrawable clipDrawable = (ClipDrawable) imageView.getDrawable();

clipDrawable.setLevel(5000);

}

В примере обрезается правая часть по центру, оставляя видимой левую часть картинки.



Чтобы оставить только четвертинку (левую верхнюю часть), изменим атрибуты

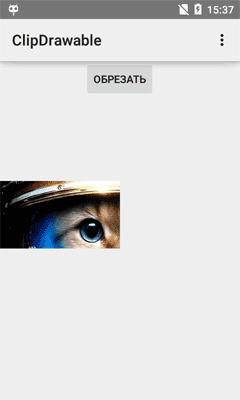
<?xml version="1.0" encoding="utf-8"?>

<clip xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/cosmos"

android:clipOrientation="horizontal|vertical"

android:gravity="left|top" />



Следующая комбинация отрежет лишнее слева и справа, оставив только центральную часть изображения.

<?xml version="1.0" encoding="utf-8"?>

<clip xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/cosmos"

android:clipOrientation="horizontal"

android:gravity="center" />

<color> (класс ColorDrawable)

Цветовые ресурсы используются в виде значений в папке **res/values**, например, в файле **colors.xml**. Но можно обращаться к цвету, как drawable-ресурсу.

android:drawable="@color/green"

<inset> (Класс InsetDrawable)

Пример создания [объёмной панели](http://developer.alexanderklimov.ru/android/design/tilemode.php#3dpanel) с помощью **inset**.

<layer-list> (класс LayerDrawable)

Можно задавать не только простые формы, но и их комбинации. Для этого служит класс **LayerDrawable**, позволяющий накладывать несколько объектов **Drawable** один поверх другого. Описав массив полупрозрачных объектов **Drawable**, вы можете создать сложную комбинацию динамических фигур и преобразований.

**LayerDrawable** описывается с помощью тега <layer-list>, внутри которого для каждого дочернего узла <item> используется атрибут **drawable**, указывающий на ресурс для наложения.

Каждый объект **Drawable** будет накладываться в соответствии со своим индексом — первый элемент массива размещается в самом низу.

Переделаем [пример программного использования LayerDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#layerdrawable) на пример с использованием XML. Создадим в папке **res/drawable** файл **layerdrawable.xml**:

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >

<item>

<bitmap

android:gravity="center\_vertical"

android:src="@drawable/blue" />

</item>

<item>

<bitmap

android:gravity="left"

android:src="@drawable/red" />

</item>

<item>

<bitmap

android:gravity="right"

android:src="@drawable/green" />

</item>

</layer-list>

Осталось применить к макету, при этом нам не нужен метод из примера по ссылке.

LinearLayout linear = (LinearLayout)findViewById(R.id.linear);

//linear.setBackground(createLayerDrawable());

linear.setBackgroundResource(R.drawable.layerdrawable); // с помощью XML

Кнопка с бликом

Рассмотрим другие примеры. Например, можно создать такую кнопку с бликом.

Layer List

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item>

<shape>

<gradient android:startColor="#339999" android:endColor="#006666" android:angle="-90.0"/>

<corners android:radius="10.0dip" />

</shape>

</item>

<item android:bottom="20dip">

<shape>

<solid android:color="#88339999"/>

<corners android:bottomRightRadius="0.1dip"

android:bottomLeftRadius="0.1dip"

android:topLeftRadius="10dip"

android:topRightRadius="10dip"/>

</shape>

</item>

</layer-list>

Марка

Можно даже создать изображение марки.



<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

<item>

<shape>

<solid android:color="#ffffff"/>

<stroke android:width="5dip" android:color="#ffffff" android:dashWidth="4dp" android:dashGap="4dp" />

<corners android:radius="4dip" />

<padding android:left="5dip" android:top="5dip" android:right="5dip" android:bottom="5dip"/>

</shape>

</item>

<item>

<shape>

<solid android:color="#FFFFE1"/>

<stroke android:width="1dip" android:color="#4A3321"/>

<padding android:left="10dip" android:top="10dip" android:right="10dip" android:bottom="10dip"/>

</shape>

</item>

<item>

<bitmap android:src="@drawable/image"/>

</item>

</layer-list>

Тени

Создав два похожих объекта и сместив их относительно друг друга, вы можете реализовать эффект тени.



<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >

<item

android:bottom="4dp"

android:right="4dp">

<shape>

<gradient

android:angle="-90.0"

android:endColor="#006666"

android:startColor="#339999" />

<corners android:radius="10dp" />

</shape>

</item>

<item

android:left="4dp"

android:top="4dp">

<shape>

<solid android:color="#88339999" />

<corners android:radius="10dp" />

</shape>

</item>

</layer-list>

Кнопка-треугольник

Создадим кнопку треугольной формы. Файл **res/drawable/arrow\_up.xml**.

<?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >

<item>

<rotate

android:fromDegrees="45"

android:toDegrees="45"

android:pivotX="-40%"

android:pivotY="87%" >

<shape

android:shape="rectangle" >

<stroke android:color="@android:color/transparent" android:width="10dp"/>

<solid

android:color="@android:color/holo\_blue\_light" />

</shape>

</rotate>

</item>

</layer-list>

Применим стиль к кнопке. Текст задавать не будем.

<Button

android:id="@+id/buttonStyle"

android:layout\_width="70dp"

android:layout\_height="70dp"

android:layout\_alignEnd="@+id/textViewInfo"

android:layout\_alignParentBottom="true"

android:layout\_marginBottom="63dp"

android:background="@drawable/arrow\_up"

android:onClick="onUpButton" />

Если понадобится такая же кнопка с направлением вниз, то достаточно добавить атрибут **android:rotation="180"**.



<transition> (класс TransitionDrawable)

Является расширением предыдущего **layer-list** и позволяет использовать наложение.

В теге **transition** указываются два объекта **Drawable**, между которыми можно переключаться с fade-эффектом и указанием продолжительности перехода.

[Пример использования](http://developer.alexanderklimov.ru/android/animation/transitiondrawable.php)

<level-list> (класс LevelListDrawable)

Используя **LevelListDrawable**, вы можете эффективно размещать ресурсы **Drawable** один поверх другого, указывая целочисленный индекс для каждого слоя.

<level-list xmlns:android="http://schemas.android.com/apk/res/android">

<item android:maxLevel="0" android:drawable="@drawable/cat\_0"/>

<item android:maxLevel="1" android:drawable="@drawable/cat\_1"/>

<item android:maxLevel="2" android:drawable="@drawable/cat\_2"/>

<item android:maxLevel="4" android:drawable="@drawable/cat\_4"/>

<item android:maxLevel="6" android:drawable="@drawable/cat\_6"/>

<item android:maxLevel="8" android:drawable="@drawable/cat\_8"/>

<item android:maxLevel="10" android:drawable="@drawable/cat\_10"/>

</level-list>

Чтобы вывести на экран определенное изображение, вызовите метод **setImageLevel()** из представления, которому назначен ресурс **LevelListDrawable**, передавая в качестве параметра индекс объекта **Drawable**, который вы хотите отобразить.

imageView.setImageLevel(5);

Представление отобразит ресурс с соответствующим (или большим) индексом. Ресурс **LevelListDrawable** нужен при компоновке виджетов.

<nine-patch> (класс NinePatchDrawable)

[Примеры использования](http://developer.alexanderklimov.ru/android/theory/draw9patch.php)

<rotate> (класс RotateDrawable)

Данный ресурс позволяет повернуть изображение на нужный градус. Допустим, у вас есть стрелки в разные стороны. Совсем не обязательно создавать копии стрелок. Достаточно нарисовать одну стрелку, а в файлах прописать нужные повороты. Например, повернём значок приложения вверх тормашками в файле **drawable/rotate\_up.xml**:

<rotate xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/ic\_launcher\_cat"

android:fromDegrees="180"

android:toDegrees="180"

android:pivotX="50%"

android:pivotY="50%" />

Присвойте атрибуту **src** компонента **ImageView** и вы увидите значок перевёрнутым.

* **drawable** — изображение для поворота
* **fromDegrees** — начальный угол поворота, в градусах. По умолчанию 0.0
* **toDegrees** — конечный угол поворота, в градусах. По умолчанию 360.0
* **pivotX** — опорная точка по X
* **pivotY** — опорная точка по Y

Опорная точка - это место для виртуальной булавки, вокруг которой будет вращаться пришпиленный листок. Для центра мы указали половину по высоте и ширине. Для неподвижной картинки углы поворота можно указать одинаковыми.

Тег **rotate** также используется в анимационных ресурсах, не путайте их.

<scale> (класс ScaleDrawable)

Позволяет масшабировать (сжать или расширить) изображение по горизонтальной (**scaleWidth**) и/или вертикальной (**scaleHeight**) оси и сместить полученное изображение в указанную часть (**scaleGravity**) доступного пространства.

<?xml version="1.0" encoding="utf-8"?>

<scale xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/ic\_launcher\_cat"

android:scaleGravity="center\_vertical|center\_horizontal"

android:scaleHeight="180%"

android:scaleWidth="80%" />

<selector> (класс StateListDrawable)

У некоторых элементов управления есть различные состояния: нажато, выбрано и т.д.. Вы можете задавать изображения для любого из таких состояний.

Чтобы описать **StateListDrawable**, создайте файл в формате XML, в котором указываются разные ресурсы **Drawable** для каждого состояния компонента. Каждый атрибут вида **android:state\_\*** может принимать одно из двух значений: *true* или *false*.

Доступны следующие состояния:

* state\_focused - в фокусе или нет
* state\_window\_focused - имеет ли фокус родительское окно
* state\_enabled - доступен или нет
* state\_checkable - может ли быть отмечен или нет
* state\_checked - отмечен или нет
* state\_selected - выделен или нет
* state\_activated - активирован или нет
* state\_pressed - нажат или не нажат
* state\_hovered - находится ли курсор над компонентом или нет (API 11 и выше)

Для списков есть также:

* state\_single
* state\_first
* state\_middle
* state\_last

Рассмотрим примеры. Нас не устраивает внешний вид кнопки. Хочется, чтобы она выглядела так (Да еще и чтобы подсвечивалась при нажатии):

Custom Button

Сделать это просто. Добавляем в ресурсы изображения нормального (button\_up.png) и нажатого (button\_down.png) состояний кнопки. И создаём в папке **drawables** XML-файл со следующим содержанием (states\_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\_down" />

<item android:drawable="@drawable/button\_up" />

</selector>

Важный момент: в списке состояний нужно сначала указывать специализированные состояния, а последним должно следовать состояние по умолчанию (без атрибутов state-\* вообще).

Ещё один момент. Кнопка может одновременно находиться в состоянии *pressed* и *focused*, поэтому сначала указывайте **state\_pressed**, а затем **state\_focused**. Иначе, если кнопка будет в состоянии *focused*, то состояние *pressed* не применится.

Теперь осталось у атрибута **background** для кнопки прописать созданный ресурс.

<Button android:layout\_width="wrap\_content" android:layout\_height="wrap\_content"

android:background="@drawable/states\_button"

android:text="Custom button"

android:textSize="18dip"

/>

Можно обойтись без использования изображений для кнопки, а воспользоваться фигурами (states\_shapes.xml).

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state\_pressed="true">

<shape>

<gradient android:type="linear" android:endColor="#6699CC" android:startColor="#99CCFF" android:angle="90.0" />

<corners android:radius="5.0dip" />

<padding android:left="20dip" android:right="20dip" android:top="7dip" android:bottom="7dip"/>

</shape>

</item>

<item>

<shape>

<gradient android:type="linear" android:endColor="#336699" android:startColor="#99CCFF" android:angle="90.0" />

<corners android:radius="5.0dip" />

<padding android:left="20dip" android:right="20dip" android:top="7dip" android:bottom="7dip"/>

</shape>

</item>

</selector>

Custom Button

А можно использовать просто цвета:

<?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"/>

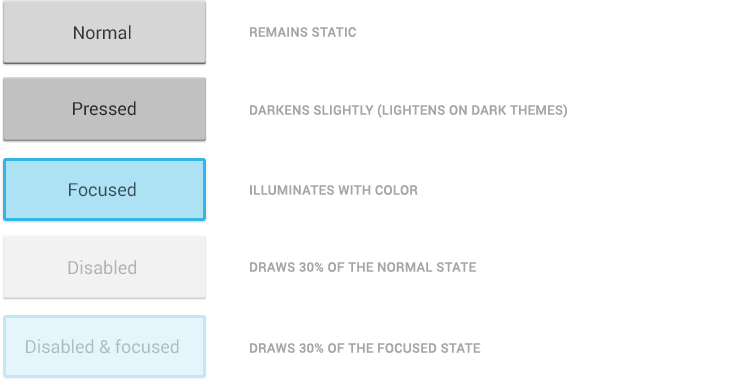
<item android:state\_focused="true" android:color="#ff0000ff"/>

<item android:color="#ff000000"/>

</selector>

Попробуйте применить файл не к атрибуту **background**, а к атрибуту **android:textColor**, чтобы цвет влиял на текст кнопки, а не на саму кнопку.

Можно использовать как шпаргалку следующую картинку.



RadioBox, CheckBox и др.

Похожим образом можно задавать стили для **RadioButtom**, **CheckBox** и т.п.. Но тут мало того, что можно отдельным ресурсом задавать фон, так отдельным же ресурсом можно задавать саму пиктограмму через атрибут button (states\_compound.xml).

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:state\_checked="true">

<shape android:shape="oval">

<gradient android:startColor="#00FF00" android:endColor="#00000000" android:type="radial" android:gradientRadius="7"/>

<size android:width="10dip" android:height="10dip" />

</shape>

</item>

<item>

<shape android:shape="oval">

<gradient android:startColor="#FF0000" android:endColor="#00000000" android:type="radial" android:gradientRadius="7"/>

<size android:width="10dip" android:height="10dip" />

</shape>

</item>

</selector>

Разметка с созданными стилями

<CheckBox android:layout\_width="wrap\_content" android:layout\_height="wrap\_content"

android:button="@drawable/states\_compound"

android:text="Item 1"

android:paddingLeft="20dip"

/>

<CheckBox android:layout\_width="wrap\_content" android:layout\_height="wrap\_content"

android:button="@drawable/states\_compound"

android:checked="true"

android:text="Item 2"

android:paddingLeft="20dip"

/>

<CheckBox android:layout\_width="wrap\_content" android:layout\_height="wrap\_content"

android:button="@drawable/states\_compound"

android:text="Item 3"

android:paddingLeft="20dip"

/>



-nodpi, -anydpi

Есть очень интересная статья о папках с квалификаторами **-nodpi, -anydpi** - [The CommonsBlog — -nodpi, -anydpi, and WTF?](https://commonsware.com/blog/2015/12/21/nodpi-anydpi-wtf.html)

Вкратце, **nodpi** для исключительных случаев, когда картинка нужного разрешения не найдётся. Например, у нас есть **res/drawable-nodpi/foo.xml** и **res/drawable-xxhdpi/foo.png**. Устройство с **-xxhdpi** должно использовать PNG; остальные - XML-версию.

Изображения в **res/drawable-anydpi/** также подойдут для любого разрешения экрана, но работают по другому принципу. Например, у нас есть **res/drawable-anydpi/foo.xml** и **res/drawable-xxhdpi/foo.png**. Все устройства должны использовать XML, даже устройства с разрешением **-xxhdpi**. Поэтому квалификатор удобно использовать совместно с другими квалификаторами. В частности, с **-v21**, что полезно при работе с вектором.

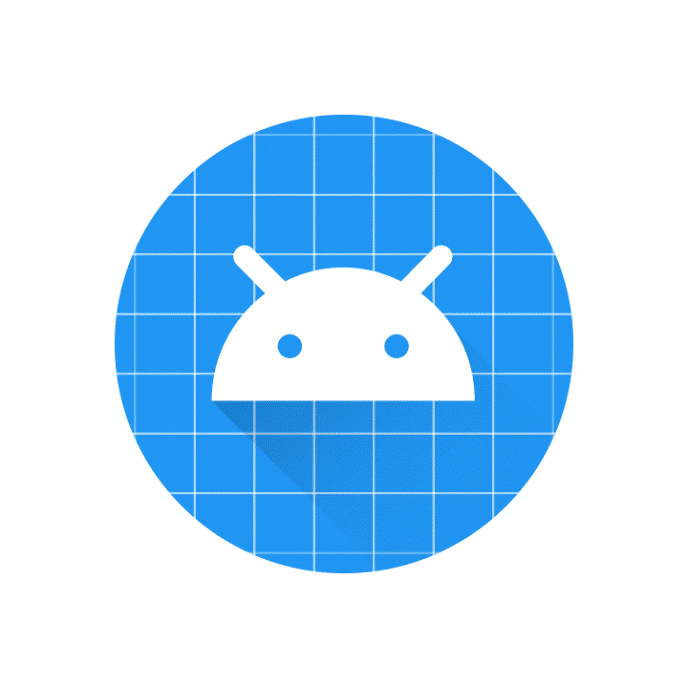
Допустим, у нас есть **res/drawable-anydpi-v21/foo.xml** и **res/drawable-xxhdpi/foo.png**. В этом случае, все устройства 5.0+ будут использовать XML. Все xxhdpi-устройства на Android 4.4 и ниже будут использовать PNG. И все остальные устройства на Android 4.4 и ниже также будут использовать PNG, подгоняя картинку под своё разрешение.

Векторные файлы нельзя размещать в папке **-nodpi**, если **minSdkVersion** меньше 21.

Папка **res/drawable/** по сути является синонимом для **res/drawable-mdpi/** и была оставлена в целях совместимости, когда на первых устройствах ещё не было деления на разрешения экрана. Никто же не предполагал, что Android так разовьётся. Сейчас эту папку используют для xml-файлов, которые ведут себя достаточно независимо то разрешения экранов при правильном использовании.

# Адаптивные значки

В Android 8.0 Oreo появилась поддержка адаптивных значков.



Для работы с адаптивными значками следует использовать API 26 и выше. Адаптивные значки состоят из двух слоёв: фон и основная часть.

В манифесте по-прежнему остаётся старая запись о значке.

<application

android:icon="@mipmap/ic\_launcher"

android:roundIcon="@mipmap/ic\_launcher\_round"

</application>

Далее следует создать файл **res/mipmap-anydpi-v26/ic\_launcher.xml**. Android Studio 3.0 генерирует подобный файл, можете изучать его.

<?xml version="1.0" encoding="utf-8"?>

<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">

<background android:drawable="@drawable/ic\_launcher\_background" />

<foreground android:drawable="@drawable/ic\_launcher\_foreground" />

</adaptive-icon>

Слои содержат векторные изображения **VectorDrawable**. К слову сказать, в качестве фона можно использовать просто цвет.

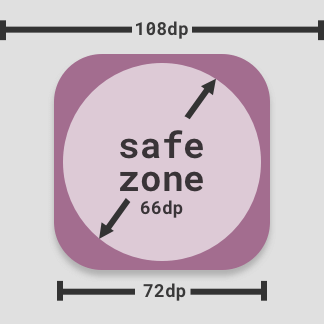
<background android:drawable="@color/something\_white"/>

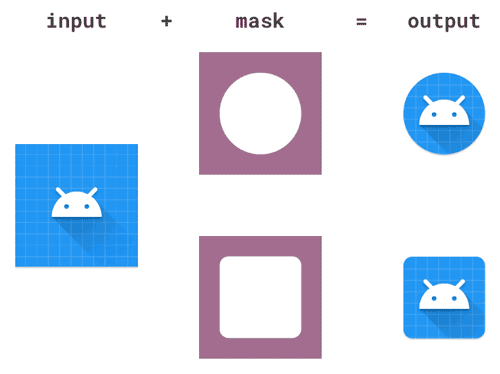
Для передней части значка можно использовать PNG-файл (используйте ресурсы **mipmap**).

Для совместимости с Android 7 вы должны также создать ещё один файл **ic\_launcher\_round.xml** с таким же содержанием.

Если изучить ресурс для фона, то можно заметить, что для значка используются размеры 108dp\*108dp. Основной слой значка имеет те же размеры, но нужно учитывать одно обстоятельство - фоновый слой работает как маска, накладываемая на передний значок. Поэтому вы должны проследить, чтобы маска случайно не закрыла важные детали значка.

Гарантировано будет виден центр значка 66dp, а 77dp применимо к общему размеру значка.





Адаптивные значки можно применить к [App Shortcut](http://developer.alexanderklimov.ru/android/shortcut.php).

На эмуляторе следует выбрать устройство Pixel и включить у него режим разработчика. Далее в настройках домашнего экрана появится пункт **Change icons shape**.

## Дополнительное чтение

[Adaptive Icons | Android Developers](https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive.html) - страница документации

[Create App Icons with Image Asset Studio | Android Studio](https://developer.android.com/studio/write/image-asset-studio.html) - создание адаптивных значков через Android Studio.

[Adapticon](https://adapticon.tooo.io/) - онлайн-просмотр эффектов некоторых адаптивных значков. Можно загрузить свой вариант - первый значок.

[Implementing Adaptive Icons – Google Developers – Medium](https://medium.com/google-developers/implementing-adaptive-icons-1e4d1795470e) - в статье приводится пример применения линейного градиента для тени.

[Designing Adaptive Icons – Google Design – Medium](https://medium.com/google-design/designing-adaptive-icons-515af294c783) - ссылки на шаблоны Sketch, Illustrator и т.д.

[Releases · nickbutcher/AdaptiveIconPlayground](https://github.com/nickbutcher/AdaptiveIconPlayground/releases) - приложение с открытым исходным кодом для удобного просмотра эффектов значков с настройками.