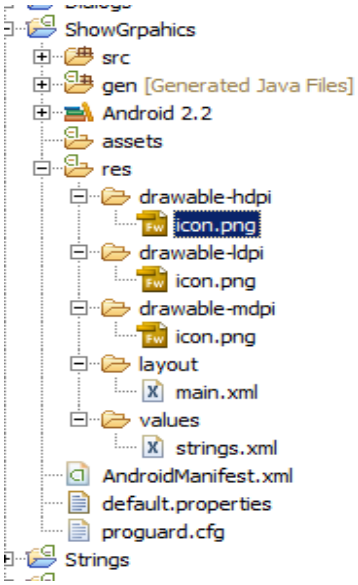
Graphics Resources in Android

Graphics can be manipulated from linked xml files.



Inside the directory res are 3 directories named drawable-* where the final part of the name refers to low, medium or high dpi.

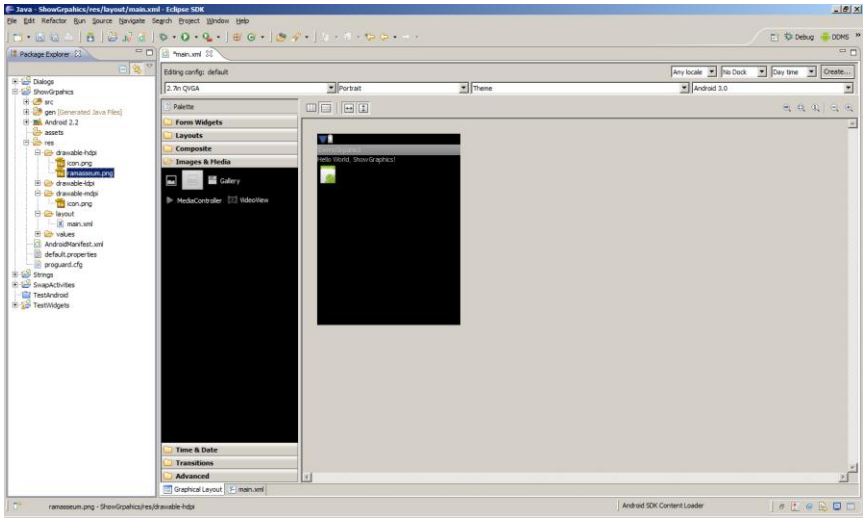
The icon included in each when the android project is created is that used for the desktop icon. This can be modified and replaced to create a unique icon for the program instead of the default android logo.

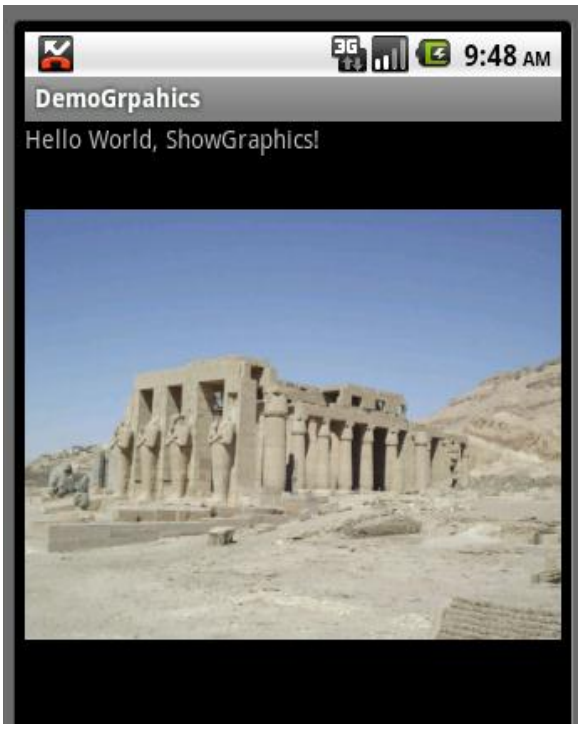
The 3 versions of the same graphic are to support android devices with differing screen resolutions. This procedure holds up to android 3.1. In 3.2 and later a differing strategy is used to handle higher resolution tablet devices.

*.png is usually the best format for graphics files. It is a lossless compression format and should be used instead of *.jpg although android will support *.jpg files.

Showing graphics

Android has an ImageView control that can be added to the xml layout for an activity.





This control can be linked to an image file. The image file must be in /res/drawable-x, the directory /res/drawable-hdpi will work with lower resolution devices. The file ramasseum.png has been copied across in the example above.

Right click on the ImageView control, select src and browse for the image to display. If src will not show up the xml can be modified directly as shown below

No java code is required.

Note that the xml code in main.xml does not look for /res/drawable-hdpi but the generic /res/drawable (which does not exist, android picks the best folder for the job)

```
<ImageView android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:id="@+id/imageView1"
android:src="@drawable/ramasseum"></ImageView>
```

Using xml to set backgounds

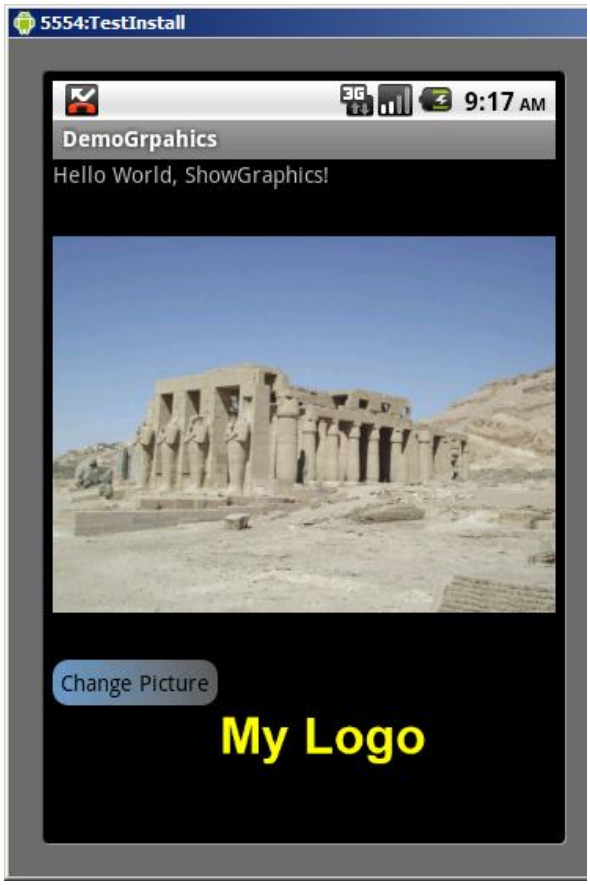
A control such as an ImageView or a Button can be set to an image or an xml file can set up a fill pattern. This will take up less memory that working with an image but the possible effects are more limited.

Create a new xml file in /res/drawable-hdpi. It does not matter what type of xml file is created as all the code will be trashed.

The code below has been designed for a Button but the xml resource is available for use in other controls.

```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android="http://schemas.android.com/apk/res/android"
     android:shape="rectangle">
     <gradient
          android:startColor="#FF6699CC"
          android:endColor="#70CCCCCC"
          android:angle="45" >
     </gradient>
     <padding
```

```
            android:left="5dp"
            android:right="5dp"
            android:top="5dp"
            android:bottom="5dp" >
        </padding>
        <corners
            android:radius="10dp" >
        </corners>
</shape>
```



- The shape refers to the shape of the image created. The following shapes are supported, rectangle, oval, line and ring.
- The gradient describes the start and end colour of the fill in hexadecimal. A program such as Photoshop or Gimp will calculate suitable values. The last 6 digits are the colour (red, green, blue) and the first 2 are the opacity. FF is a solid colour 00 is fully transparent
- The padding describes the edge around the fill, dp is device pixels
- Corners allows a rounded corner effect with the stated number of degrees.

The new xml is linked to the object within main.xml. This can be by typing in a line such as `<Button android:layout_height="wrap_content" android:id="@+id/bChangePicc" android:layout_width="wrap_content" android:text="Change Picture" android:background="@drawable/fill"></Button>` or by selecting properties_background and browsing for the file. Here the file has been named fill.xml and is saved in /res/drawable-hdpi.

Changing images at run time

The image src of the ImageView can also be changed at run time by an event. An appropriate image must still exist in /res/drawable-hdpi for the code to work.

Here pressing a Button will cause the image displayed within the ImageView to change. The application interface will need a Button and a setOnClickListener handler to recognise that it has been pressed.

The ImageView will also need to be referenced in code (within onCreate)

```
picBox = (ImageView)findViewById(R.id.imageView1);
```

When the Button is clicked the ImageResource of the ImageView can be changed:

```
public void onClick(View v) {
          picBox.setImageResource(R.drawable.ramfront);

}
```

To swap between 2 (or more images) a counter (for a set) or boolean (for a pair) will need to be referenced. Here changed has been declared as a boolean variable

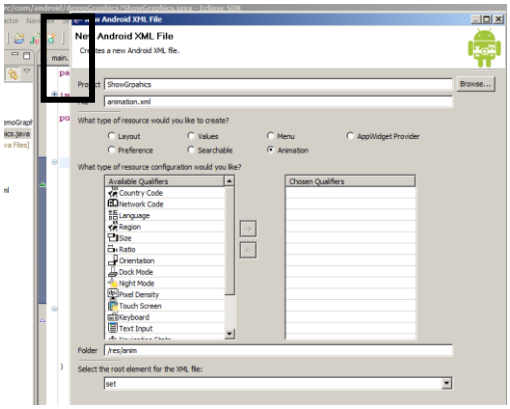
```
 if (changed == false)
{
      picBox.setImageResource(R.drawable.ramfront);
      changed=true;
}
```

Frame by frame animations

This is the animation used in cartoons where a sequence of images is presented in turn to simulate a single moving object

- The images will need to exist and be placed in /res/drawable-hdpi
- There must be an ImageView to host the images
- An xml file is used to control the image order and timing of the animation.

The new concept here is the xml file that will describe the animation. Use the xml wizard within Eclipse to create a new xml file of the type animation. The file name does not matter but must folllow the usual naming conventions and end in .xml. Leave the root element as set.

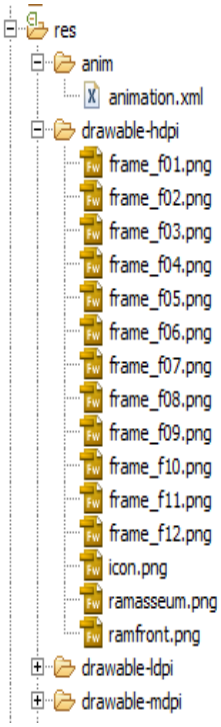


The following outline xml code will be created.

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

Here is the modified code to describe animating 2 files

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android">
     <item android:drawable="@drawable/frame_f01"
android:duration="30" />
     <item android:drawable="@drawable/frame_f02"
android:duration="30" />
</animation-list>
```



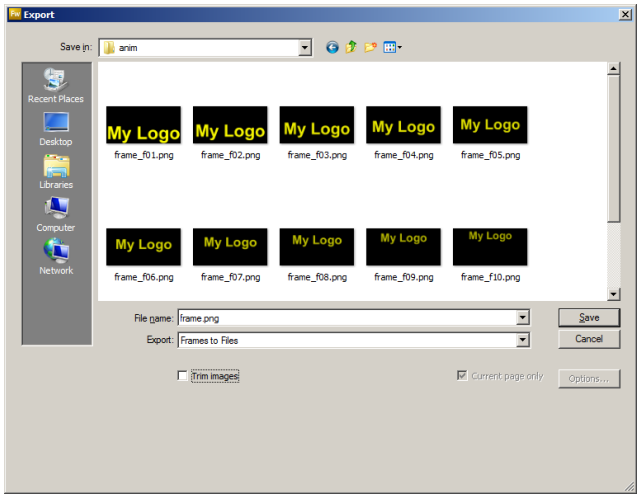
`frame_f01` is the name of the 1st image without its file extension. That image is a *.png and is stored in /res/drawable-hdpi.
`android:duration="30"` is the time that the image will be displayed for in milliseconds. Adding further similar lines will handle more images within the animation sequence.

The animation will run indefinitely. To make the animation run only once add this setting `android:oneshot="true"` to `<animation-list>`

The images all need to saved in /res/drawable-hdpi
but the xml code that will control the animation is in /res/anim

Creating sets of animation images in Fireworks

- Open a new Fireworks project and ensure that the image size is appropriate for the android screen. 200 by 100 pixels is small enough for many screens.
- Avoid a transparent background but set the canvas colour to that of the Activity background colour.
- Create a simple animation.
- Choose file export, frames to files. Ensure that ‘trim images’ is not selected and that each image is saved as a png. Fireworks will try to export gif files and android does not like these.



Referencing the images in code

An ImageView needs to be added to the Activity and referenced in the java code. An AnimationDrawable object is also required to link to the animation xml file

These objects need to be declared:

```
ImageView picAnim;
AnimationDrawable anim;
```

The ImageView needs to be instantiated and an onClickListener set up in onCreate

```
picAnim = (ImageView)findViewById(R.id.imgAnim);
picAnim.setBackgroundResource(R.anim.animation);
picAnim.setOnClickListener(this);
```

The click event can then be handled in onClick

```
if (v== findViewById(R.id.imgAnim))
{
          anim = (AnimationDrawable) picAnim.getBackground();
          anim.start();
}
```

The Image view still maintains its default android icon. If we try to remove this in code by setting an ImageResource:
`picAnim.setImageResource(R.drawable.frame_f01);`
The animation messages will be ignored.

The best plan is to remove the reference to the default icon within main.xml by deleting this code from the ImageView reference.

```
android:src="@drawable/icon"
```

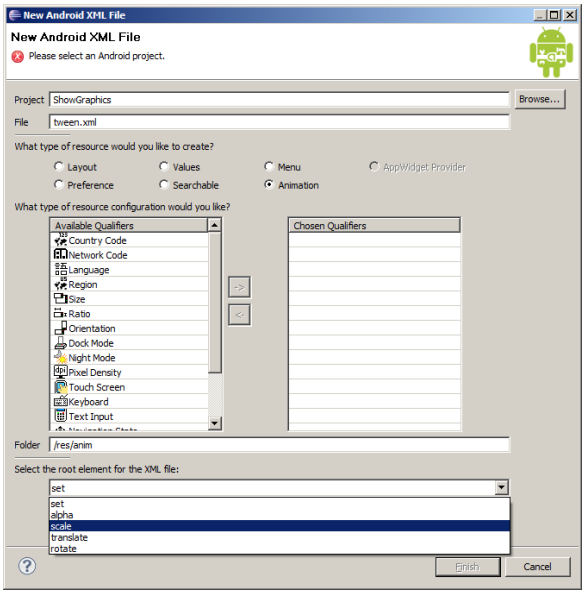
The animation (if it has been set to loop continuously) can be stopped in code with `anim.stop();`

To use code to check if the animation is running (so that it can be stopped and started at run time) use `anim.isRunning()` this will equate to true or false.

Animating with tweening

This technique takes a start and end state of the animation and fills in the gaps (the in-betweens) to create an animation effect.

A new xml file is required to describe the animation effects. The root element chosen determines how the tween will behave. Here a scale effect has been selected.



The xml file below will scale the x and y positions of an image from ½ size to full size (it could be set to larger sizes such as 2.0).

The scaling will be based around or pivot on the centre (50% of the original x and y)

The animation will last for 1000 ms (1 second)

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

If the xml settings above are not correct the xml file may parse and the project may build but the animation will not work. The numerical values can, however, be altered within reasonable parameters.

The new xml file is referenced and started in the java code as follows:

```
Animation tween =
AnimationUtils.loadAnimation(ShowGraphics.this, R.anim.tween);
picBox.startAnimation(tween);
```

The code will need to be linked to a suitable event.

Note the following naming choices:

- The Animation object is named tween but could be given any reasonable name
- ShowGraphics is the name of the java class (the Activity) that the code is running in
- picBox is the name of the object being tweened. This is an ImageView but other objects such as a Button can be tweened.

Multiple tweening effects

The xml file responsible for the tween can be modified to handle more than 1 tween effect. The example below may not be the best aesthetic effect but shows what can be done. Here the target of the tween will grow larger and rotate:

```
<?xml version="1.0" encoding="utf-8" ?>
<set
xmlns:android="http://schemas.android.com/apk/res/android">
     <scale
          android:fromXScale="0.5"
          android:toXScale="1.0"
          android:fromYScale="0.5"
          android:toYScale="1.0"
          android:pivotX="50%"
          android:pivotY="50%"
          android:duration="1000" >
     </scale>
     <rotate
          android:fromDegrees="0"
          android:toDegrees="360"
          android:pivotX="50%"
          android:pivotY="50%"
          android:duration="500"
          android:startOffset="500" >
     </rotate>
</set>
```

The existing <scale> code is unaltered except that it is now within a <set> parameter. The android namespace `xmlns:android="http://schemas.android.com/apk/res/android` has been moved up from <scale> to the new root element <set>.

An additional element <rotate> has been created.

- Start from 0^o and rotate to 360^o
- Pivot around the centre (50% x and y) like the scale
- Run for 500 milliseconds

- Do not begin until the scale tween has run for 500 milliseconds (the offset)

To have the animation run at a slowly increasing speed change the <set> parameter to

```
<set
xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">
```

This code brings the accelerate_interpolator into play.