

# android design guidelines

version 1

February 2011



# table of contents

|                                 |    |
|---------------------------------|----|
| introduction .....              | 3  |
| sizes and resolution .....      | 5  |
| UI elements .....               | 7  |
| icons .....                     | 13 |
| dialog and listview icons ..... | 23 |
| widgets.....                    | 24 |
| draw9patch .....                | 27 |
| gestures .....                  | 30 |
| gingerbread .....               | 30 |

# introduction

## This is a Programmer's life we are just living in it.

When Doug Bowman, former creative Director for Google resigned, he posted a quote on his blog that in my opinion sums Google up perfectly.

*“Yes, it’s true that a team at Google couldn’t decide between two blues, so they’re testing 41 shades between each blue to see which one performs better.”*

Android is the creation of technicians and analytical minds, super genius programmers who someday will probably unlock the secret to creating human life with code, modern day Robin Hoods who give mobile carriers an open source doorway to the kingdom of apps.

But apple gets them because of their unclear philosophy of design.

Android is crushingly programmer heavy. In my opinion, they have not given enough consideration to asset creation or creative direction. With apple, the moment you start designing an app, you know how they want you to build it. Their design plans are so complete that you can immediately access it. I often dream that my house is an iPhone app; it has become comfortable. With Android, there is a false sense of freedom to do whatever you want. Unfortunately, when you start taking all the various screen sizes and resolutions into account, it doesn't realistically pan out that way. As a designer, I like to create the whole picture. I look at designing apps as making a little painting that I will carve up into pieces for another to make into a collage. Android thwarts this. To successfully design an Android app, the designer should tinker on the little things and let development sort out the rest.

Designing an Android app needs to be a constant collaboration between design and development. The look and feel of the app should be discussed by both parties from the beginning. Additionally, app creators should target specific phones. There is too much trial and error involved in asset creation to be left squarely in the hands of a designer and this can lead to resource problems. Since many assets can be created with XML, the design responsibility falls to both parties. Once a look and feel is decided upon, the designer would be smart to take a tell-me-what-you-need-and-I'll-make-it approach.

An Android app can look as lush and stylized as an iPhone app with the right amount of planning. The process to achieve this will be much slower as assets go through the crucible of multiple resolutions (and functionality is placed on the racks for the various devices). It may take a bigger budget and softer touch, but it is possible and worthwhile to make a great looking app.

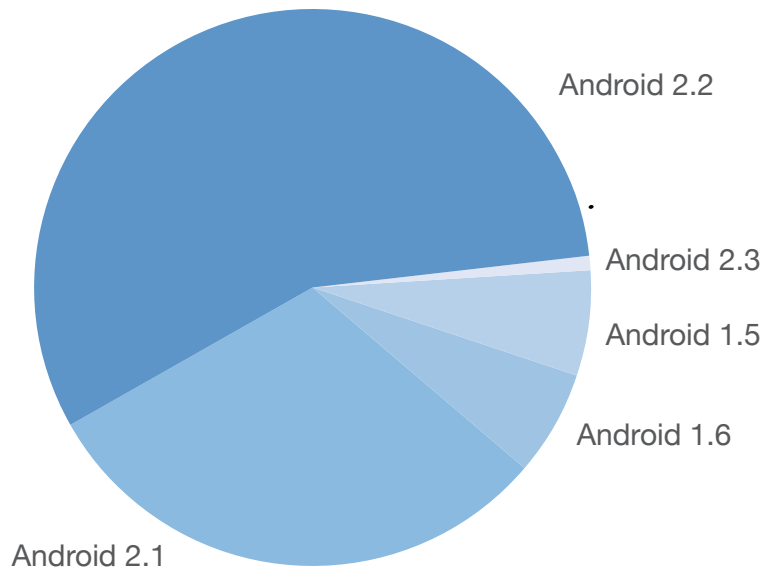
## It is more than just removing the back button.

As a designer, my comfort comes from a sense of designing what I know. This is why when I started designing apps, the first thing that came to mind were websites. It makes sense. There are enough similarities. Both rely on buckets of information. My very first app design was extremely poor, relying on drop-down menus and an ill-placed home button. This strategy turned out to be very poor and it made for a project that immediately needed to be retooled. It should have been obvious that since the iPhone uses completely different UI elements and conventions that the web approach would not work. With this wisdom in mind, it is baffling that people are now porting iPhone apps to the Android, keeping them consistent with their iPhone brethren and blatantly disregarding the functionality and feature set native to Android. It is incorrect to consider an app to be portable from iPhone to Android. The differences are many and what principles and elements may work great for the iPhone don't speak to Android peculiarities or functionality. Many of the differences are programmer-centric, however, asset management must be given a completely different treatment to do the device justice, as well.

I was working on a "port" of an established iPhone app, and I learned the error of my ways. The assets that I made were consistent with the iPhone. This turned out to be a headache for the developers and inevitably a set back for the client. What I took away from this is that even if a look and feel is established, the moving pieces need to be built from the ground up to accommodate the phone. In the Android world, we need to define a boundary between buttons and icons, reassess what needs to be designed and what is better left programmed, and learn to remain open-minded, patient and exploratory. The Android is a curious frontier and while there are definitely rules and standards, they are not always obvious

As a disclaimer, I am a designer, and while my knowledge of programming has increased a fraction while creating a document, it should be known that Android makes it difficult for anyone who is not a programming to decipher their rules. So with that in mind, understand that while my research has been abundant, this document should be considered a designer's translation of Android and therefore subject to error.

While it certainly is not my place to set a standard for Google, in this document I intend to at least create a guideline for us to follow.



*This chart displays the Current Distribution of operating systems as collected at the beginning of February 2011*

## Know your market.

Worldwide, there are over 90 Android devices that run the gambit of operating systems. While there are still a good amount of devices running Android 1.6 and below, this document is more specifically directed at Android operating system 2.1 and on.

You will notice that Android 1.6 is fading out. Unless a client specifically asks for the application to be designed for an earlier platform, it should be presumed that we are developing for 2.1 and on.

In future versions of this document, I will go into detail about designing for 1.6 and earlier. There are some noticeable differences. For instance, icons are handled completely differently than is mentioned in this document.

| Platform    | API Level | Distribution |
|-------------|-----------|--------------|
| Android 1.5 | 3         | 3.9%         |
| Android 1.6 | 4         | 6.3%         |
| Android 2.1 | 7         | 31.4%        |
| Android 2.2 | 8         | 57.6%        |
| Android 2.3 | 9         | 0.8%         |

# sizes and resolution

## Designing for multiple screen resolutions and sizes

The source of 90% of Android design woes come from the multiple resolution and screen sizes. If assets are not created properly, they can create a heartbreaking adventure in iterations. With that ill-fated port that I mentioned earlier, while the assets looked fine on most devices, on a few they appeared dithered and blurry. What made it so confounding was the lack of a clear indication as to what I did wrong, so my only solution was to try and try again.

The fact that you are designing for multiple resolutions should be in the front of your mind through the entire design.

### This should weigh into the consideration of

- what your buttons look like
- what sort of gradients you use
- how complex your icons are
- what sort of backgrounds you make. If you make one at all, as a lot of this can be handled better by a developer.

## Generalized Screen Sizes and Resolutions.

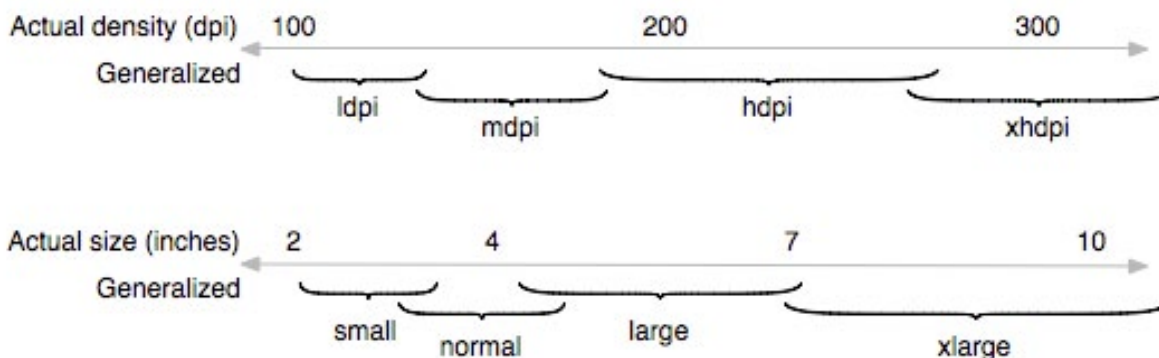
Due to the widely varying array of devices, it is next to impossible to pinpoint the specific resolution that you should be designing for, with this in mind Android has charted four generalized resolutions and four generalized densities. It breaks down like this.

**There are four generalized sizes.**

- Small (2-3 inches)
- Normal (3-5 inches)
- Large (4-7 inches)
- X-Large (7-10 inches) - tablets only

**And four generalized Resolutions**

- LDPI (100-120 dpi)
- MDPI (120-160 dpi)
- HDPI (160-240 dpi)
- XHDPI (240-320 dpi)



Regardless of the actual screen size or density, applications are programed into these four categories.

|                           | Low density (120), <i>ldpi</i>           | Medium density (160), <i>mdpi</i>        |
|---------------------------|--|--|
| <i>Small screen</i>       | QVGA (240x320)                           |  |
| <i>Normal screen</i>      | WQVGA400 (240x400)<br>WQVGA432 (240x432) | HVGA (320x480)                           |
| <i>Large screen</i>       |  | WVGA800* (480x800)<br>WVGA854* (480x854) |
| <i>Extra Large screen</i> |  |  |

|                           | High density (240), <i>hdpi</i>        | Extra high density (320), <i>xhdpi</i> |
|---------------------------|--|--|
| <i>Small screen</i>       |  |  |
| <i>Normal screen</i>      | WVGA800 (480x800)<br>WVGA854 (480x854) |  |
| <i>Large screen</i>       |  |  |
| <i>Extra Large screen</i> |  |  |

For the most part, screen sizes and densities correlate.

## How This Effects Layout.

When creating wireframes for an Android app, it is probably wise to work your layout into multiple sizes to make sure that your application will work across as many platforms as possible. Obviously, this will make for a longer project, but the due diligence will make for a better application. It may not be the best course to base your design exclusively on the top devices - especially based on who your audience is.

## How This Effects Design.

### **You have two choices.**

Keep the design simple and squeaky clean. Don't use intense gradients and consult your developer about the background theme. Or be prepared to make custom assets for each of the various resolutions and screen sizes. If you want to make that fancy background, do it, but be prepared to make a ton of assets.

### **Bottom Line.**

When designing for Android, not taking the complexity of density and screen size into consideration will make the project more difficult. I repeat; bring development into the process early on to test the art, layout and elements before completion.

# UI elements

For some reason Google has remained vague about any sort of sizing or rules for their UI elements. While I respect the lack of limitations, when you consider the various densities and screen sizes, it would make sense to establish at least a loose guideline. I have spent countless hours googling terms such as "Awesome Android Tab" and "Customized Menu". I took a myriad of screenshots and measured the various navigation elements and after days of research, I have come up with something. I don't want to call it a standard, because that it is a bit misleading. Instead, I will call it a guideline.

## The Tab Bar

I don't know exactly what to say about the Tab Bar. We work with them everyday. They serve the same purpose with Android as they do with the iPhone. With Android, they can be top or bottom aligned.

There is a standard-ish Android Tab Bar, but it is somewhat unsightly. The prettier apps usually opt to make their own.



*The standard tab bar.*

## The Consistent Size.

None of my research has turned up a pixel size for the tab bar. Not a standard, not a regular, not a suggested. So after measuring a sampling of screens, I have come up with my own standard or “consistent”.

### Consistent Android Tab Bar sizes

- HDPI - 480x96
- MDPI - 320x64
- LDPI - 240x48

## It Came from Cupertino

Perhaps too editorial for this document, but, Android applications that try to look like iPhone applications don't work so well. I would argue that in the instance of the Android, the Tab Bar belongs on top for a couple of reasons. Functionally, the menu comes from the bottom, obscuring the tab bar. Additionally, the hierarchy of tasks and activities is set up completely differently, using option and contextual menus.

It begs the question -“Who are you are making the app for?” iPhone navigation is most likely not as intuitive for seasoned Android users, since these two user groups generally don't overlap.

## The Options Menu.

The option menu stores activities. From here, you will be able to reach settings, save, logout, etc.

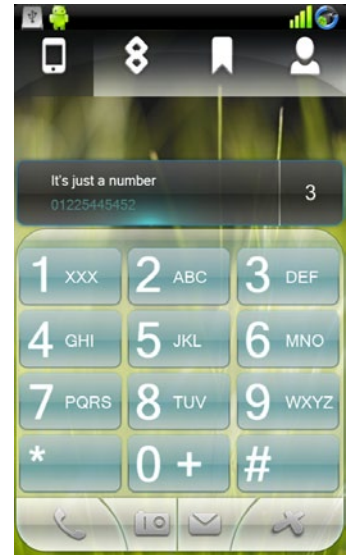
To compare it to the iPhone, it contains what you would find in the navigation bar, mixed with what you would find in an action sheet. It is somewhat customizable. It can be skinned but the size will not change. The width is adjustable based on the number of buttons and the size of the screen.

### The Consistent Height

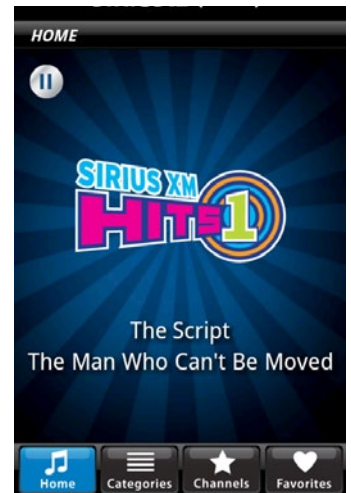
- HDPI - 100x
- MDPI - 66x
- LDPI - 50x

## The Importance of being an Options Menu.

The importance of the options menu cannot be understated. The complete openness of the Android operating system demands its need. All of the functionality that would rest in a nav bar, toolbar or action sheet on the iPhone should exist within the options menu. The inclination, when creating a design for an Android app based on an iPhone app is to maintain the semblance of a nav bar. This should be discouraged.



*Slick and transparent*



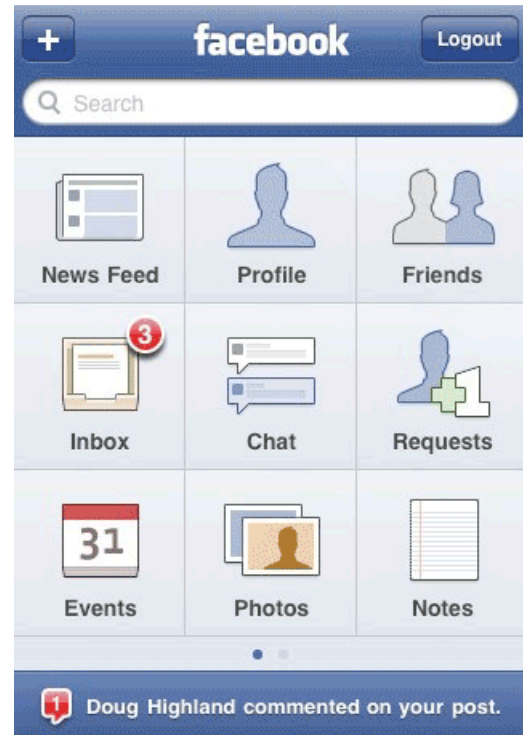
*Not so much*







VS.



This Facebook screen is a tiny 240x320. And yes, you could argue that there are buttons at the top of the screen that resemble a toolbar. But if you consider the additional functionality Facebook offers on the iPhone; the ability to logout, to select favorites, etc, it becomes clear that presenting that much functionality would be unwieldy and messy. Thus, the options menu.

When designing an Android app, consider hiding all functions that edit your current screen in the options menu.

## The Context Menu.

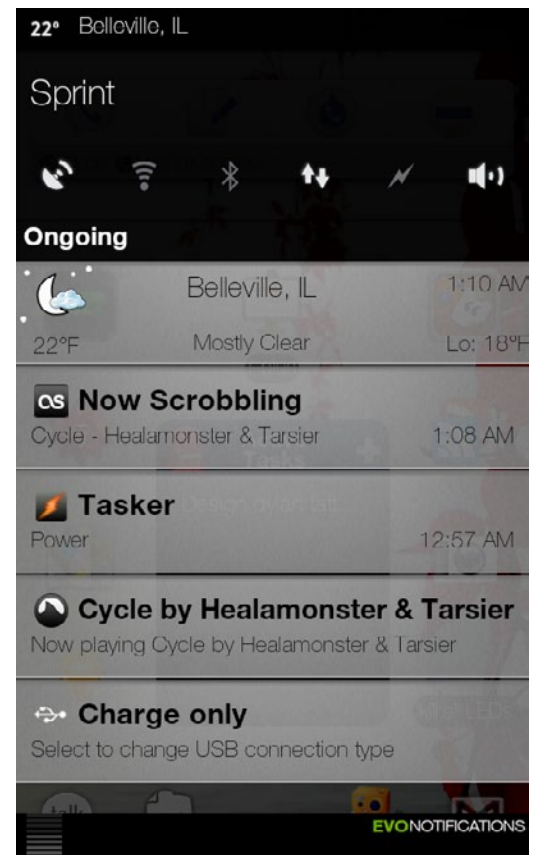
The Context Menu is similar to the right click on the desktop. The user will touch and hold to bring up the menu which will provide commands that pertain to the selected activity.

In email, for example, touching and holding a particular email will bring up a context menu to delete or archive the email.

They are customizable, but this should be a conversation between design and development as far as the worth of customization.

**While the width of the cells are adjustable, the standard height is;**

- HDPI: 100 px
- MDPI: 66 px
- LDPI: 50 px



*Sleek, Pretty, Hidden*

## When to use an Option Menu. When to use a Context Menu.

Selecting the right kind of menu can be a bit confusing. To sum it up, any activity that is global to the app should either go in the options menu or the list view. Activities that pertain solely to the content of a cell in the list view would summon a context menu. Option menus contain activities such as composing an email or logging out of an application. Context menus contain activities such as deleting a specific email, viewing or editing a contact or sending a text message to the specific contact.

## Prioritizing Operations.

Due to the often limited screen heights it is important to place most frequently used operations first. For example, if you have a search function in your Option Menu, (this is where search belongs, by the way) it should most likely be the first option available. Settings is considered another high priority function in the Option Menu.

## No Context Menu is a good Context Menu.

In some instances, it is unavoidable, such as with your contact book, where you need to have a slew of options attributed to each field of the list view, however, if you can avoid using a context menu, it is advisable that you do so. As the Context Menu lacks any physical representation, it is not intuitive to the user. Android suggests duplication of functionality in some instances, such as the contact book, where the user can get to the phone number by tapping and holding the contact and by tapping the phone number in the list view. Use the context menu for advanced user functions.

## Short Names in the Option Menu.

Much like the iPhone Springboard, The Android option menu will truncate long names. So keep it short.

## Paper beats Rock. Dialogue Beats Option Menu.

When a Dialog box is being displayed, it is assumed that the Menu button is disabled. A Dialog box is usually something that is important and must be handled before global functions should continue.

## Dim or Hide.

There are times when an item that is in the option or contextual menu will not pertain to the context at hand. Android's example is the forward button, which obviously doesn't work until after the back button is pressed. If you have an instance like this in the options menu, dim it out. If you have an option like this in the contextual menu, hide it completely.

## Dialog Boxes.

Unlike the alerts in iOS, the Android Dialog box is customizable. It can bear any theme and its size is adjustable to the content. As usual, be careful with this often unnecessary customization.

**In Android, a dialog box is used for a whole slew of functions such as:**

- searching
- alerts
- progress
- status bars,
- color wheels
- date pickers

When a dialog box is up, most of the functionality of the app is disabled, including the search bar. So when designing with dialog boxes in mind, consider if it is worth the disabled functionality.

The category of dialog contains a variety of different types of modals.

## Alert Dialog

This type is similar to the alert on the iPhone, however, It can take a couple of different forms. An alert Dialog is used to display a warning, a text message, or a choice (such as quitting and application).

An alert can either have up to three buttons or be a list of selectable items, usually displaying check boxes or radio buttons.

## Progress Alert.

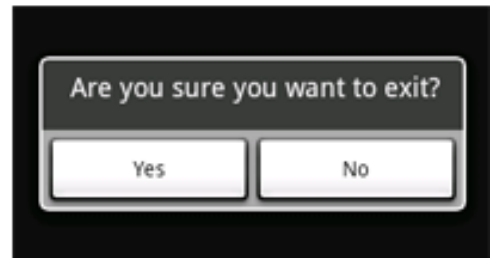
The progress alert comes in two forms.

The spinning wheel represents an undefined progress and the box will be present until its function is complete

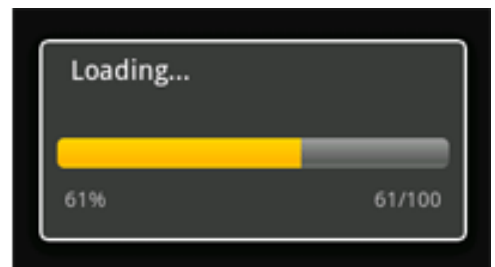
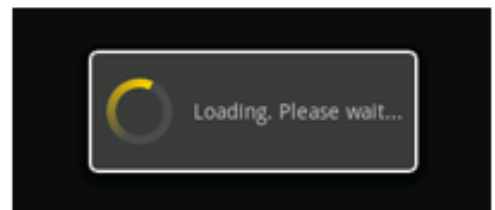
The progress bar conveys to the user a set amount of time or activities that need to occur before a task is complete.



*A Custom Color Picker*



*Examples of Alerts*



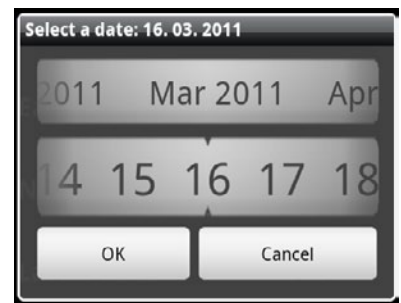
## Android's fantastic what-you-see-is-what-you-get date and time picker.

Android's date and time picker are not much to look at. Certainly they are sparse, utilitarian and functional, but also uninspiring and dull.

It is quite possible to come up with a new rendition of this bland dialog. That should be a discussion between designer and developer as creating custom dialogs with this much functionality can be an expensive and time consuming venture.

In the case of a short time line or smaller budget, this is what you've got. Enjoy it!

Here is an example of a custom date picker. Not fantastic, however it is a vast improvement on the standard.



## List Views

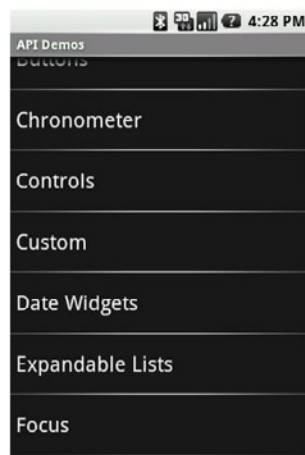
Customizing list views can be a bit tricky. In development, list views are transparent fixtures over the default background (the dark # #FF1919). By default, list views have a faded edged gradient at the top of the screen. While this effect is flashy and neat looking, initially it caused all sorts of problems for drawing performance.

To counter this, Android came up with a script cache color hint. What this does is set RGB color by default to the background values.

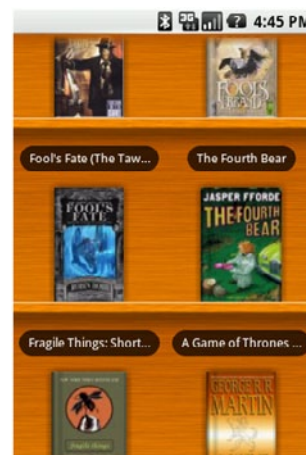
Unfortunately, this has terrible results when the user swipes through a list on a custom background.

To avoid this wretched effect, it should be noted in the document for developers that the cache color hint should be disabled. It is considered disabled if set to #000000, thereby transparent.

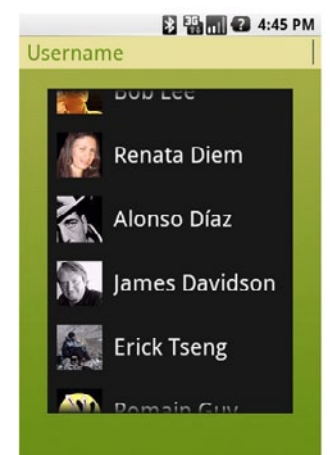
It is important to note, however, that without the cache color optimization, the effect on performance can be an issue. Therefore, some consideration about the utility being performed is important to keep in mind.



Run of the mill List View



A custom List done right.



Custom List View Botch Job

# icons

## Icons vs Buttons

I will share the most important lesson that I learned from that ill-fated iPhone port. In Android, there is a clear distinction between an icon and a button. They need to be treated as different assets. With the iPhone, we balance between Tab Bar icons and custom widgets that include labeled buttons, however with Android, these buttons need to be considered two assets. The reason for this is that Android assets need to be draw9patched in order to accommodate that various screen sizes (see draw9patching).

Android icons can be any shape because they sit in a square bounding box the are essentially gridded out. Later in the document, we will discuss Android standard bounding sizes, but at the moment let's look into creating custom elements and how to create correct custom bounding boxes.

When creating custom Android navigation, such as a tab bar, it is important to consider the icon to be the button and the tab bar itself to be a background. Gradients can not be easily draw9patched without losing their finesse.

For example:



This is your tab bar. It is created for HDPI so its resolution comes in at 480x72. This element can be used in development without draw9patching. It should be considered a background without any functionality.

This is your icon. It should be roughly 48x48 before effects. The area that the Icon takes up should be 1/3 of your tab bar.

- So your bounding box should be 160 px wide and 75 px high. The red box indicates this.
- The yellow box indicates the layers of effects. In this instance, the icon has a 2 pixel outer glow and a 2 pixel drop shadow.
- The Blue box indicates the asset itself. In this instance it is 48x48

When slicing the asset, the PNG should be created with the bounding box taken into consideration. In this instance, even though the icon is 48x48, the PNG needs to be 160x72. This will make draw9patching much easier.



## Icons, Icons, Icons

Android is particular about icons and has set a fairly rigid guideline on icon creation. The next section is a summary of Android's icon guidelines from their developer website.

Android is designed to run on a variety of devices that offer a range of screen sizes and resolutions. When you design the icons for your application, it's important to keep in mind that your application may be installed on any of those devices.

It is necessary to design a set of icons for each of the screen densities. Below is a chart of the standard sizes of each type of icon.

Also, since most buttons need to be draw9patched, it is important to consider that the icon is a separate asset from the button itself. All Icons should be saved as a transparent PNG.

| Icon Type                          | Standard Asset Sizes (in Pixels), for Generalized Screen Densities |   |   |
|------------------------------------|--|---|---|
|                                    | Low density screen ( <i>ldpi</i> )                                 | Medium density screen ( <i>mdpi</i> )       | High density screen ( <i>hdpi</i> )         |
| Launcher                           | 36 x 36 px   | 48 x 48 px                                  | 72 x 72 px                                  |
| Menu                               | 36 x 36 px   | 48 x 48 px                                  | 72 x 72 px                                  |
| Status Bar (Android 2.3 and later) | 12w x 19h px<br>(preferred, width may vary)                        | 16w x 25h px<br>(preferred, width may vary) | 24w x 38h px<br>(preferred, width may vary) |
| Status Bar (Android 2.2 and below) | 19 x 19 px   | 25 x 25 px                                  | 38 x 38 px                                  |
| Tab                                | 24 x 24 px   | 32 x 32 px                                  | 48 x 48 px                                  |
| Dialog                             | 24 x 24 px   | 32 x 32 px                                  | 48 x 48 px                                  |
| List View                          | 24 x 24 px   | 32 x 32 px                                  | 48 x 48 px                                  |

## Launcher Icons

Much like the App Icon for the iPhone, the Android app is activated with the Launcher Icon. The user opens the Launcher by touching the icon at the bottom of the Home screen, or by using any hardware navigation controls, such as a trackball or d-pad. The Launcher opens and exposes the icons for all of the installed applications.

## Android 2.0

With Android 2.0, launcher icons are recommended to be front-facing, rather than the three-quarter perspective of earlier operating systems.

## The Standards and Styles.

When it comes to designing the launcher, Android has a sur-



Standard Android Icons

prisingly high number of rules. While no one is going to frown upon you if you break these rules, this is what Android expects out of a launch icon.

According to the Android's Guidelines, launcher icons should be modern, clean and contemporary. They should not appear aged and should avoid overused symbolic metaphor. They should be simple and iconic.

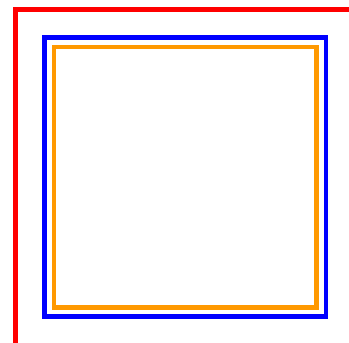
The Android icon is caricatural in nature. Simple and exaggerated so that they are clear on the smallest resolutions. They should be geometric and organic and most importantly, textured. Additionally, they should be top-lit.

## Adventures in Bounding Box pt. 1; The Launcher Icon

The reason that Android icons can be practically any shape has to do with the fact that their icon falls on a grid. To make an icon correctly, these rules must be followed. This applies to every icon you make. The figure on the following page displays how an Android icon is set up.

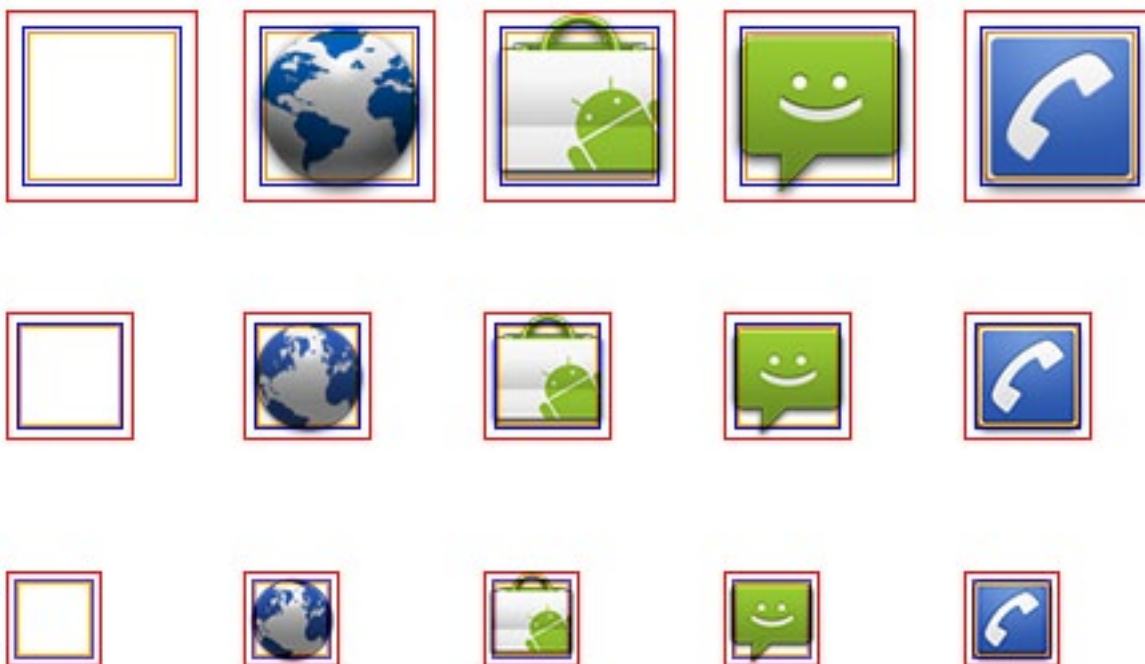
### In this instance, it is the launcher, but it applies to all icons.

- The red box is the bounding box for the full asset.
- The blue box is the bounding box for the actual icon of any shape.
- The orange box is the recommended bounding box for the actual icon when the content is square.



*Learn to love this box*

The box for square icons is smaller than that for other icons to establish a consistent visual weight across the two types.



The sizes are as follows:

### Launcher icon dimensions for high-density (HPDI) screens:

- Full Asset: 72 x 72 px
- Icon: 60 x 60 px
- Square Icon: 56 x 56 px

### Launcher icon dimensions for medium-density (MPDI) screens:

- Full Asset: 48 x 48 px
- Icon: 40 x 40 px
- Square Icon: 38 x 38 px

### Launcher icon dimensions for low-density (LDPI) screens:

- Full Asset: 36 x 36 px
- Icon: 30 x 30 px
- Square Icon: 28 x 28 px

## Texture and Color

To be consistent with Android's standard, the launcher icon should appear tactile and consist of primary colors. If you look at the examples, you will see that they usually combine two neutral colors in high contrast. Saturated colors do not tend to look good on the Android springboard. Android has given some examples of colors and textures that do well for launcher icons. The examples are on the next page.

All of these textures can be found at [http://developer.Android.com/guide/practices/ui\\_guidelines/icon\\_design.html#templatespack](http://developer.Android.com/guide/practices/ui_guidelines/icon_design.html#templatespack)



## The Launcher Icon Drop Shadow Effect

In order to keep your icon consistent with the others on the spring board. A very specific drop shadow should be used. Below are the drop shadow amounts for photoshop and illustrator at all screen resolutions.

### Photoshop

- HDPI: #000000 75% opacity, Distance=1.5 Size=4.5px, Angle=90
- MDPI: #000000 75% opacity, Distance=1 Size=3px, Angle=90
- LDPI: #000000 75% opacity, Distance=.75 Size=2.25px, Angle=90

### Illustrator:

- HDPI: Multiply, 75% opacity, x=0, y=1.5, Blur=4.5px
- MDPI: Multiply, 75% opacity, x=0, y=1, Blur=3px
- LDPI: Multiply, 75% opacity, x=0, y=.75, Blur=2.25





#FFFFFF



#BFBFBF



#6699FF



#99CC33



#FFCC00



#808080



#3366CC



#00CC00



#FF9900



#000000



#404040



#003399



#669900

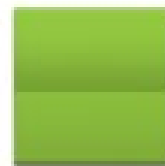
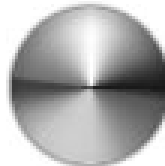
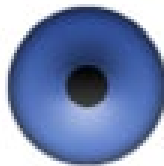
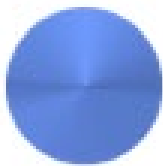
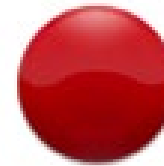
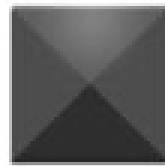
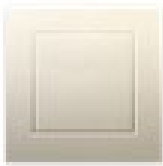
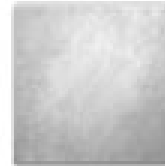
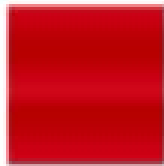


#FF6600



#CC0000

*Android Supplied Examples of good colors and textures*



## Menu Icons.

The menu icons are used in the option menu, accessible by pushing the menu button. Unlike, the tab icon, there is no need to design two states, one icon will do. Because the options menu is a uniform color, it is recommended that your icon remain monochrome, preferably gray. Since the icon will need to be draw9patched, it should be saved as a transparent PNG.

## The Gingerbread Conundrum

We have already discussed how three sets of assets must be created to account for the various DPIs. Now with Android 2.3, another variation has been established.

With Gingerbread, Android is introducing a whole new level of UX in order to establish as much of a standard as possible and because of this, it handles menu icons differently. In designing icons, you will find that icons for 2.2 and below will appear inverted in color on 2.3.

## Designing Menu Icons for 2.3

**With Gingerbread, there are a couple of changes that need be noted;**

- Icons have a larger safe frame; icon content is smaller within the full asset. Final asset sizes have not changed.
- The color palette is slightly lighter.
- No outer glow effects are applied.
- Menu icons can now be rendered on either dark or light backgrounds.

The following guidelines describe how to design menu icons for Android 2.3 (API Level 9) and later.

## Adventures in Bounding Box pt. 2; The menu icon

Android menu icons can be any shape just so long as they fit into their bounding boxes. Since the menu bar is a fixed size, it is probably a good idea to use the Android standard sizes for your icon.

**To reiterate;**

- The Red Box is the full asset
- The Blue Box is the recommended bounding box for the actual icon
- The Orange box is the bounding box for a square icon.



### Menu icon dimensions for high-density (HPDI) screens:

- Full Asset: 72 x 72 px
- Icon: 48 x 48 px
- Square Icon: 44 x 44 px

### Menu icon dimensions for medium-density (MDPI) screens:

- Full Asset: 48 x 48 px
- Icon: 32 x 32 px
- Square Icon: 30 x 30 px

### Menu icon dimensions for low-density (LDPI) screens:

- Full Asset: 36 x 36 px
- Icon: 24 x 24 px
- Square Icon: 22 x 22 px

To keep consistent with the Android standard, menu icons should be flat, face forward, and remain grayscale.

## Menu Effects

Listed below are the specifications of effects in order to keep your menu icon consistent with the Android standard:

### Corner rounding: when appropriate

- HDPI - 3 px corner radius
- MDPI - 2 px corner radius
- LDPI - 1.5 corner radius

### Gradient:

- 90°, from #8C8C8C to #B2B2B2

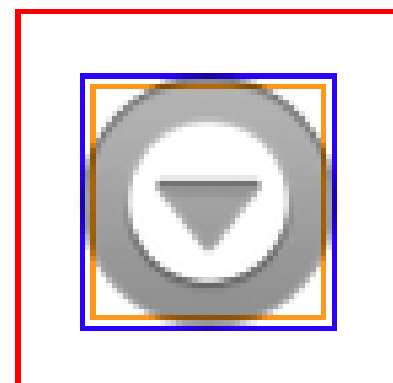
The Following effects are for Photoshop only at Medium Dpi.

### Inner shadow:

- #000000, 20% opacity
- angle 90°
- distance 2 px
- size 2 px

### Inner bevel:

- depth 1%
- direction down
- size 0 px
- angle 90°
- altitude 10°
- highlight #ffffff, 70% opacity
- shadow #000000, 25% opacity





Examples of menu icons

## Android 2.2 and Earlier

While there are not a whole lot of differences, there are enough to take notice.

### All icons for 2.2 and earlier require a slight pixel safe frame

- HDPI: 48X48, 6px Safe Frame
- MDPI: 32X32, 4px Safe Frame
- LDPI: 24X24, 3px Safe Frame

## Effects

Android suggests that you create the icon in Illustrator and then bring over to Photoshop for effects.

Menu icons are flat and front facing. A slight deboss and some other effects, which are shown below, are used to create depth.

### Light, effects, and shadows for launcher icons.

- 1 - Front part: Use fill gradient from primary color palette
- 2 - Inner shadow: black | 20 % opacity; angle 90° | distance 2 px; size 2 px
- 3 - Outer glow: white | 55% opacity; spread 10% | size 3 px
- 4 - Inner bevel: depth 1% | direction down size 0px; angle 90° | altitude 10°; highlight white 70% opacity; shadow black 25% opacity

### Color palette

- White; r 255 | g 255 | b 255; Used for outer glow and bevel highlight.
- Fill gradient; 1: r 163 | g 163 | b 163; 2: r 120 | g 120 | b 120; Used as color fill.
- Black; r 0 | g 0 | b 0; Used for inner shadow and bevel shadow.



## Tab Icons

There are few differences between the menu icon and the tab icon, except that with the tab icon, two assets need to be created to differentiation between active and inactive.

## Adventures in Bounding Box pt. 3; The Tab Icon

As we have already discussed the reason for bounding boxes, I will merely present to you the sizes.



### Tab icon dimensions for high-density (HDPI) screens:

- Full Asset: 48 x 48 px
- Icon: 42 x 42 px

### Tab icon dimensions for medium-density (MDPI) screens:

- Full Asset: 32 x 32 px
- Icon: 28 x 28 px

### Tab icon dimensions for low-density (LDPI) screens:

- Full Asset: 24 x 24 px
- Icon: 22 x 22 px

## Tab Icon color and effects

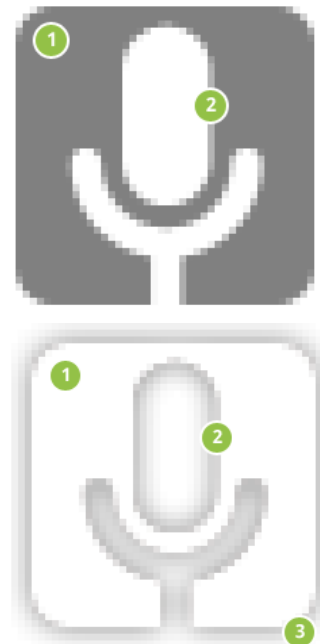
Tab Icons should be matte and forward facing.

### Inactive

- Fill Color #808080
- The inner content should be subtracted and left transparent in the PNG.

### Active

- Fill Color #FFFFFF
- The Inner Content should be subtracted and let transparent in the PNG.
- Outer Glow. #000000, 25% opacity 3 px.



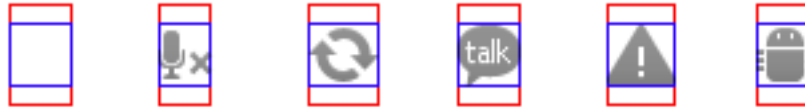
## Status Bar Icons

The Status Bar icon is used to represent notifications from your app. Status bar icons have changed with Gingerbread, so it will be important to create assets, not only just for the different densities, but also for the different operating systems.

## Android 2.3

Status bar icons are tiny and should be made using simple shapes and forms.

## Adventures in Bounding Box pt. 4; The Status Bar Icon



### Status icon dimensions for high-density (HDPI) screens:

- Full Asset: 24 x 38 px
- Icon: 24 x 24 px

### Status icon dimensions for medium-density (MDPI) screens:

- Full Asset: 16 x 25 px
- Icon: 16 x 16 px

### Status icon dimensions for low-density (LDPI) screens:

- Full Asset: 12 x 19 px
- Icon: 12 x 12 px

## Status Icon Effect

### Fill gradient:

- 90°, from #828282 to #919191

### Inner shadow:

- #FFFFFF, 10% opacity
- angle 90°
- distance 1px
- size 0px

To get this effect in Illustrator, the icon should be scaled up and the effect should be expanded.



### Inner content:

- Inner content should be subtracted and left transparent in the PNG.

## Android 2.2 and Earlier

In earlier operating systems, the status bar icon is boxier and set at 25 x 25 with a two pixel safe frame. They should have corners rounded by 2 pixels.

Rounded corners must always be applied to the base shape and to the details of a status bar icon shown in the figure below.

## Status Bar Effects

Status bar icons should be high contrast and face forward. Due to their size, it is advisable to work with the effects in photoshop.

### 1 - Front part:

- Use fill gradient from primary color palette

### 2 - Inner bevel:

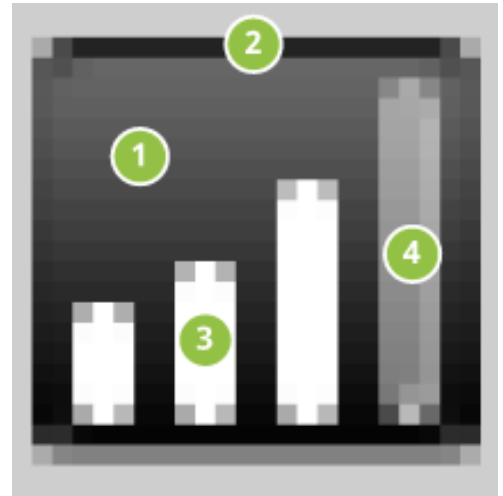
- depth 100% | direction down
- size 0px | angle 90° |
- altitude 30°
- highlight white 75% opacity
- shadow black 75% opacity

### 3 - Detail:

- white

### 4 - Disabled detail:

- grey gradient from palette
- + inner bevel: smooth | depth 1% |
- direction down | size 0 px | angle 117° |
- altitude 42° | highlight white 70% | no shadow



## Color palette

Only status bar icons related to the phone function use full color; all other status bar icons should remain monochromatic.

# dialog and listview icons

Unlike the iPhone alert, the Android dialog is customizable. Here are some guidelines for building dialog icons. Dialog and listview icons are pretty much the same except that Android's convention is to give the listview icon a slight inner shadow rather than a drop shadow.

## Sizes

All dialog icons have a 1 pixel safe frame

### Dialog icon dimensions for high-density (HPDI) screens:

- Icon: 48 x 48 px

### Dialog icon dimensions for medium-density (MPDI) screens:

- Icon: 32 x 32 px

## Dialog icon dimensions for low-density (LDPI) screens:

- Icon: 24 x 24 px

## Dialog and Listview Effects

These are the effects for an icon made for a standard Android Alert. They have a light gradient and a slight drop shadow.

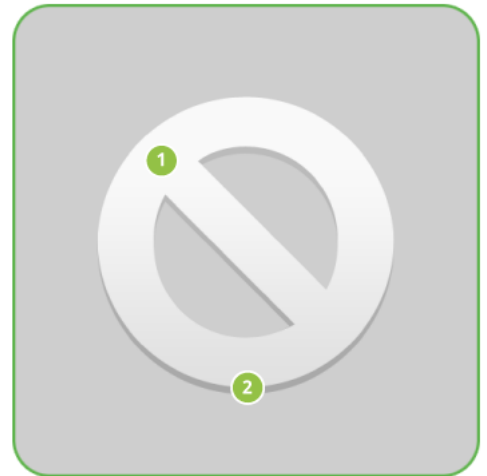
### Dialog Icon

#### 1 - Front part: gradient

- overlay | angle 90°
- bottom: r 223 | g 223 | b 223
- top: r 249 | g 249 | b 249
- bottom color location: 0%
- top color location: 75%

#### 2 - Inner Shadow:

- black | 25% opacity
- angle -90°
- distance 1 px
- size 0 px



### Listview Icon

#### 1 - Inner shadow:

- black | 57 % opacity
- angle 120°
- blend mode: normal
- distance 1 px
- size 1 px

#### 2 - Background:

- black | standard system color

These icons are displayed in list views only.



# widgets

One of the most interesting things that the Android delivers is the widget (not to be confused with a widget, which can be anything that triggers functionality). The widget is like a mini application extension of the app that runs on the home screen of the Android. The widget displays the applications most relevant information at a quick glance. Users pick the widgets they want to display on their Home screens by touching & holding an empty area of the Home screen, selecting Widgets from the menu, and then selecting the widget they want.



The widget has three main controls a bounding box, a frame, and the widget's graphical controls and other elements. The important thing to note, is that a widget has ONE function. The function maybe playing music, telling time or informing you of new Google voice messages. The potential for different widget uses is immense, as they allow the app to multi task in clear sight without being activated.

## Smaller is Better

Because a widget is going to remain on the Home screen, it is important to try to create the clearest display of information in the smallest size possible so not to become a nuisance to the user.

## Designing a Widget.

Select a bounding box size for your widget.

All widgets must fit within the bounding box of one of the six supported sizes, or better yet, within a pair of portrait and landscape orientation sizes. This is so your widget looks good when the user switches screen orientations.

## Three States for your Widget Buttons.

If your widget has any toggle functionality, (such as a music player,) make sure that the buttons have three states: inactive, pressed and active.



## Widget Sizes

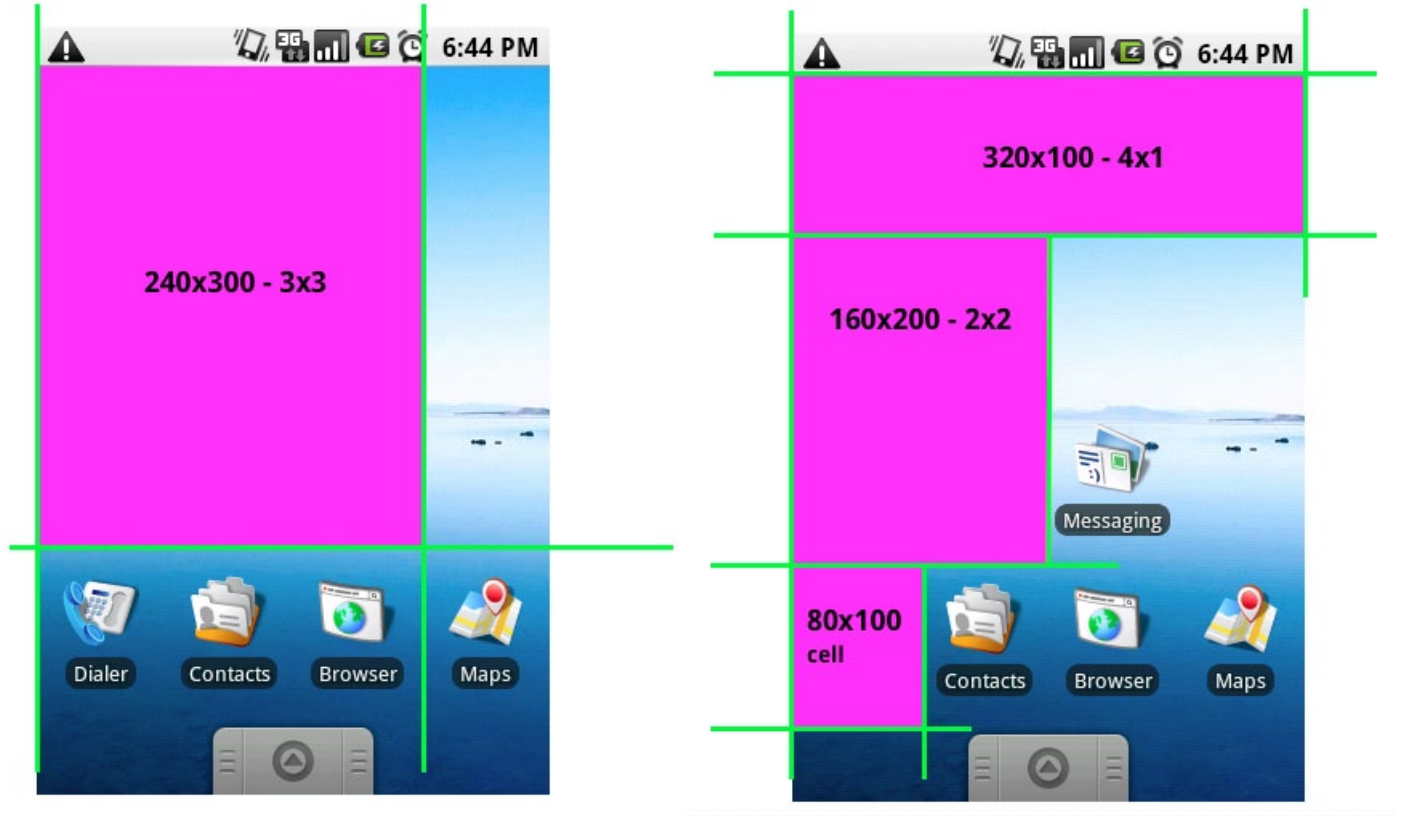
The Android Home Screen is based on a screen grid of 4 x 4 and these correspond to the dimensions of the widget bounding boxes. Make sure your content does not extend to the edges of the dimensions, rather that it is framed in the bounding box. Widgets can be skinned, but it might be wise to use the standard Android templates to at least frame your functionality. These templates can be found at [http://developer.Android.com/guide/practices/ui\\_guidelines/widget\\_design.html#file](http://developer.Android.com/guide/practices/ui_guidelines/widget_design.html#file)



*Everyone loves telling time*

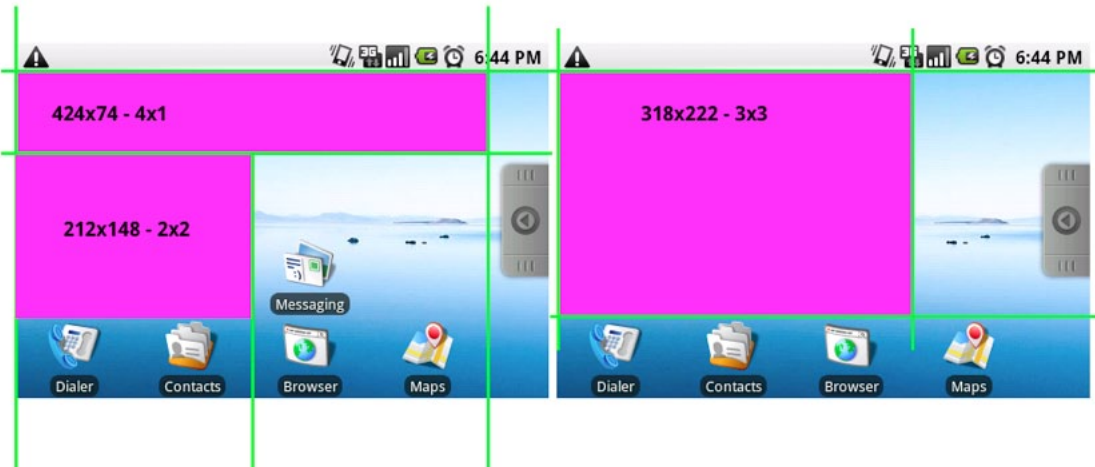
In Portrait Mode, each cell is 80 x 100 pixels. With this in mind the standard widget sizes are

- Cells      Pixels
- 4 x 1      320 x 100
- 3 x 3      240 x 300
- 2 x 2      160 x 200



In Landscape mode, the each cell is 106 x 74. The standard widget sizes are;

- Cells      Pixels
- 4 x 1      320 x 100
- 3 x 3      240 x 300
- 2 x 2      160 x 200



Keep in mind that these sizes are only Android's suggestions and there are some instances of Widgets that take up more cell space. The MixZing widget, for example is a popular dynamic music playing widget that takes up 4 x 4, in other words the whole screen.

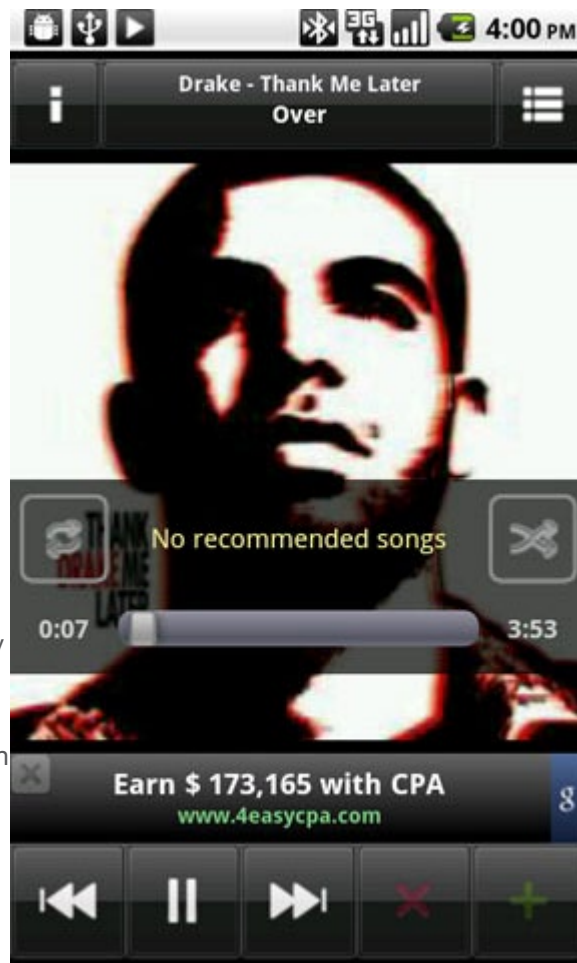
## Widget Hassles.

Like most graphic elements for the Android, widgets have their sets of complications as well. For instance, most backgrounds will have to be draw9patched. With this in mind, make sure that the corners of the bounding box have the minimal amount of gradient possible.

Also, on devices with a low pixel depth, graphics have the propensity to dither and band. This is something that can be fixed by developers through a "proxy" drawable.

As with most complicated graphics, this should be a conversation between artist and developer on finding the best approach.

As we move forward with Android apps, we should consider how we can play with the widget concept. What sole functionality do user's want from a given app and how do we present it so it is constantly captivating.



Full Screen Widget

## draw9patch

Due to the multitude of devices and resolutions, certain assets need to be draw9patched. Draw9patching, or 9-slicing is the action of selecting certain portions of a PNG that will be allowed to stretch and expand, leaving the rest of the image intact. You should create draw9patched assets if they are solid colored, such as a button or tab, or if they are on a transparency, such as an icon. You should never attempt to draw9patch a complex image, such as one that contains effects or one that has a rich gradient, as the complexity of the image will certainly be compromised.

This is an example of an asset that should be draw9patched



This is an asset that absolutely should not.



The program used to draw9patch can be obtained by downloading the Android SDK from their developer website at <http://developer.Android.com/sdk/index.html> and should be by any designer taking on an Android project. Draw9patching takes some experimentation. In my searches, I found very few how to guides and settled on a youtube video by a designer with a mumbling problem. I will attempt to explain the process on the next page.



Figure 1

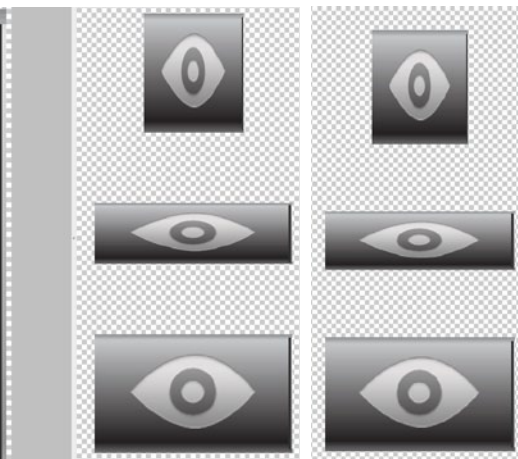


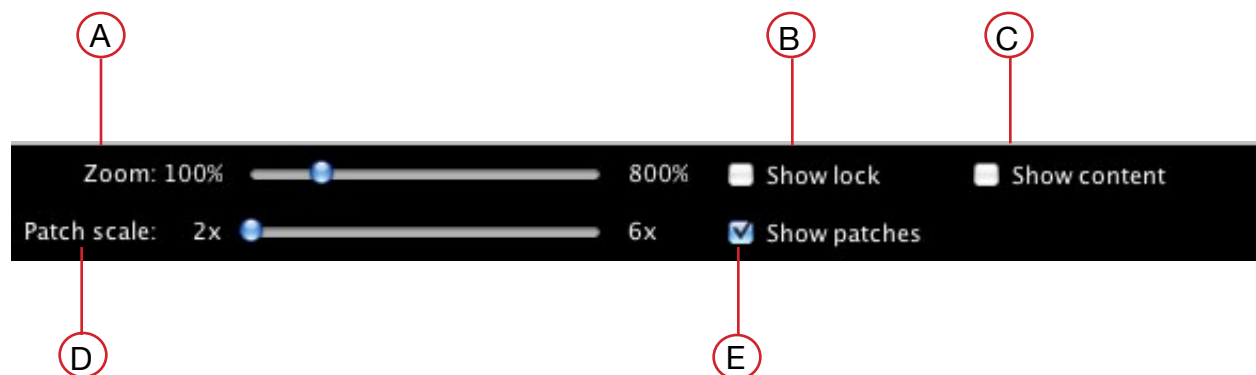
Figure 2.

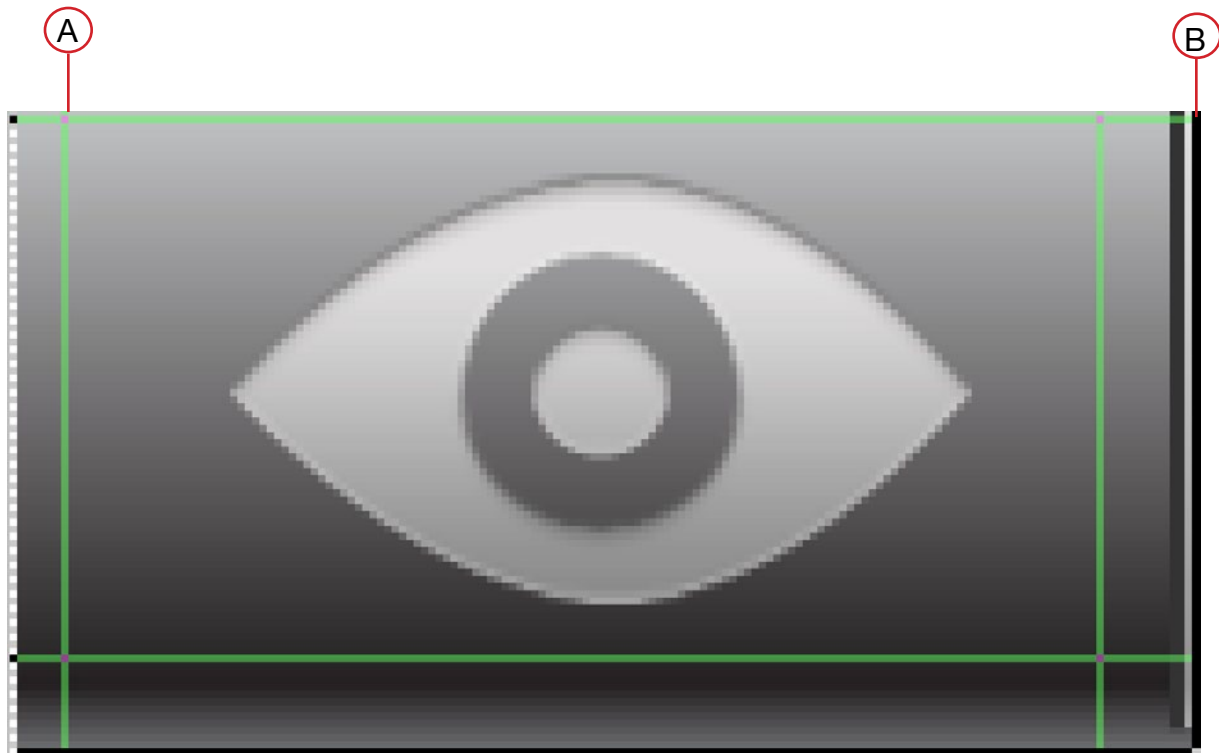
This is what the draw9patch tool looks like. I have intentionally chosen a PNG that should not be sliced to emphasize when this procedure should not be done. To get started, drag the PNG into the work area.

The right pane of Figure 1 displays what an asset looks like when it is 9patched and scaled. In Figure 2, the PNG has not been scaled yet. If the assets is used without draw9patching, this is how the image will appear on different devices.

**These are the very simple tools to the draw9patch program. Not a terrible amount of options.**

- A - The slider will zoom into your main work space.
- B - Show Lock will display the nondrawable area of the graphic on mouse over.
- C - This will highlight the content area in the preview images.
- D - Patch Scale will display how your image looks at different scales.
- E - Show patches will show you the stretchable patches in your main work space.





When you drag your image into draw9patch, it will create a 1-pixel border around your image. This pixel space will be the area in which you draw your patches.

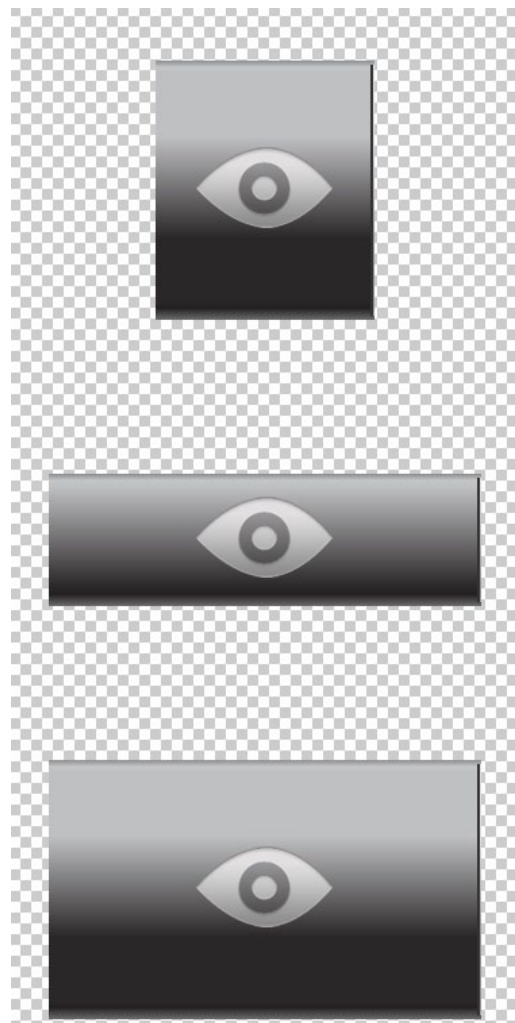
- A - Click your mouse on the top and left side to choose the areas that will be stretchable. In this image everything inside the green box will remain untouched regardless of the dimension of the device and only the frame itself will stretch. Left click the stretch pins to erase
- B - Click and drag your mouse on the right and bottom side to select the area of the image that will be affected by the draw9patching.

This takes a good amount of experimentation to do right. And unfortunately, even if you think it looks perfect, it will very likely take a couple rounds of conversation with the developer to get the asset right.

If you have draw9patched it correctly, the preview of the asset will display something like this.

Now, as I said earlier, gradients do not stretch well. The figure to the right displays what happens, reinforcing the notion that icons and buttons must be considered different assets.

When you save your PNG the file type will change to a 9.PNG. This is the is deliverable file.

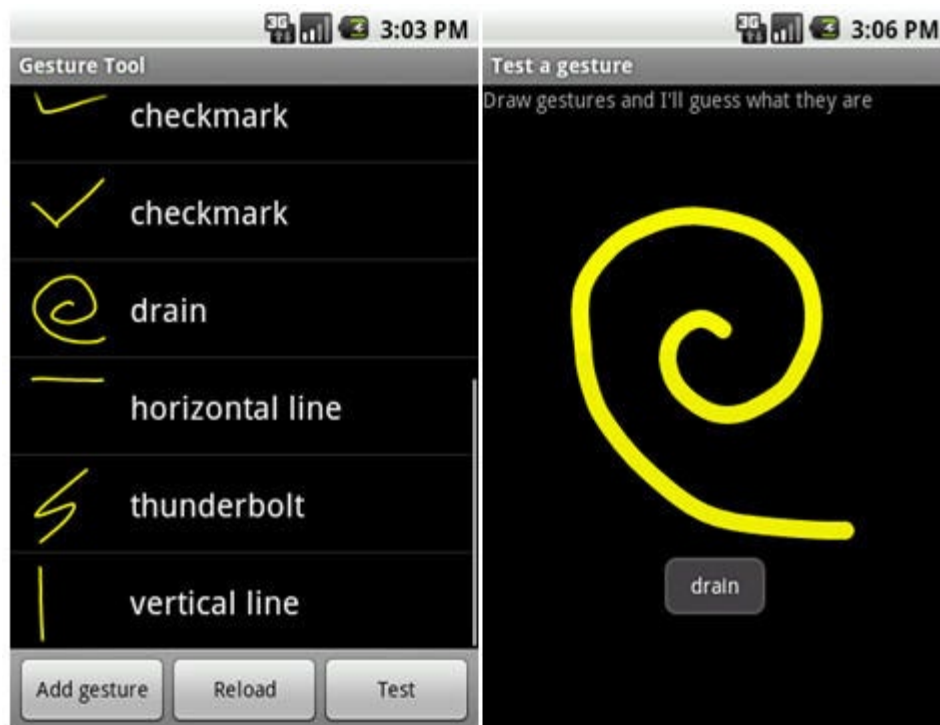


# gestures

After hours wracking my brain against the illuminati-esque coder speak of the Android Developer Guidelines and trying to research the scope of Android Gestures, I, your humble narrator, have decided that I need to speak to a developer to gain insight. Aside from the standard tap, double tap and tap and hold, it would appear that you can program any sort of gesture, such as a spiral or a squiggly line.

This is probably immensely awesome. If only my little designer brain could comprehend it.

Look forward to reading all about this in Version 2.



*Every App deserves a thunderbolt gesture.*

# gingerbread

With Android 2.3, Google made some improvements/changes to its operating system. Most of them are not design related, but good to know about anyway.

## Application Management

Gingerbread introduces a much stronger policing system to watch apps that drain the battery and shut them down when they are using too much power and running in the background. Additionally, it comes with a task manager tool that reports exactly which resources are being consumed by which apps and force stop any application.

## Updated UX

Although it is not as redesigned as promised, there are still a couple of UI changes;

- Simplified Color; Google Version 2.3's "simplified color scheme" includes a darker notification bar, cleaner status bar icons and black based menus.
- A Fancy New Keyboard; With Gingerbread, Google has redesigned the keyboard and claims that it is faster and more intuitive. It includes a built in dictionary as well as a user dictionary, An improved auto correct and speech to text functionality. It also supports multi touch.
- Improved Cut and Paste; Additionally, they have improved the cut and paste functionality. It simply requires a long press on website or text field to copy text to the clipboard.

## New Gadgets.

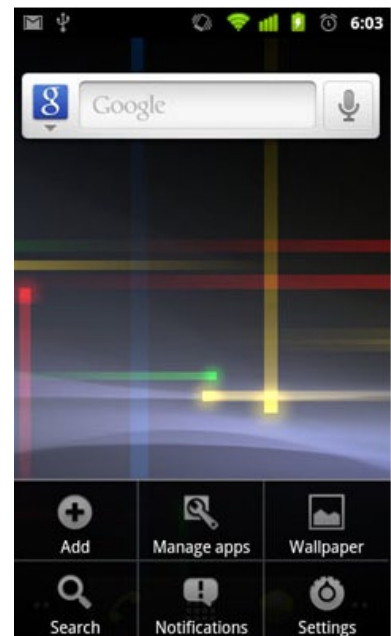
The Gingerbread operating system supports several new technologies that upcoming Android phones will offer.

**This includes;**

- Gingerbread will offer support of near field communication. Soon users will be able to tap their devices against NFC sensors in order to exchange information (eventually including credit card payments)
- A whole bunch of new sensors, including gyroscopes, gravity sensors, even barometers.
- Gingerbread also supports internet calling. Be advised, the carriers have to grant permission.
- New development tools to help design high-end video games.



Android 2.2



Android 2.3

