mutualmobile

android design guidelines

version 1.1

April 2011

table of contents

introduction

The discipline of Android design

In many ways, designing an Android application is the same as designing for any other mobile app. Android applications follow the same user experience rules that all mobile applications should follow:

- Know your audience.
- Simplify your functionality when you can, keep it organized and neat when you cannot.
- Keep it intuitive and user friendly, understanding that you have mere seconds to win or lose a user.

If there is one thing that complicates designing for Android, it is in the nature of dealing with open source software. It is commendable that Android was written for this intent - existing on any mobile device without exclusivity. For a designer though, it creates a situation where due diligence is necessary. If you jump into designing an Android application without educating yourself first, you will find yourself in a frustrating tug-of-war of trial and error with your developer. But, by learning the structure of the operating system, having a basic understanding of how assets are used and what the naming conventions should be, it is absolutely possible to create a stunning app.

Know your place

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, specific phones and operating systems should be targeted to keep expectations in line. Since many assets can be created with XML, the design responsibility actually falls to both parties. Once a look and feel is decided upon, the smart designer takes a tell-me-what-you-need-and-I'll-make-it approach of collaboration.

An Android app can look as lush and stylized as an iPhone app with the right amount of planning. The process to achieve this, however will most likely be slower since assets need to be created for multiple resolutions, and functionality must be added and removed for specific devices.

It is more than just removing the back button

Android is its own culture with its own complexities and functionality perks. Too often apps are ported directly from iPhone to Android without considering the larger landscape. It is incorrect to consider an app portable from iPhone to Android. The differences are many and those principles and elements which work great for the iPhone don't necessarily speak to Android peculiarities or functionality. Many of the differences are programmer-centric, however, asset management and user flows must be given a completely different treatment to do the device justice.

I was working on a “port” of an established iPhone app and learned the error of my ways. The designs that I made were iPhone centric, as to be consistent with the app that they already had. This turned out to be a headache for the developers and inevitably a set back for the client. What I took away from this experience is that even if a look and feel is established, the moving pieces need to be built from the ground up to accommodate the operating system. 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 remain open-minded, patient and exploratory.

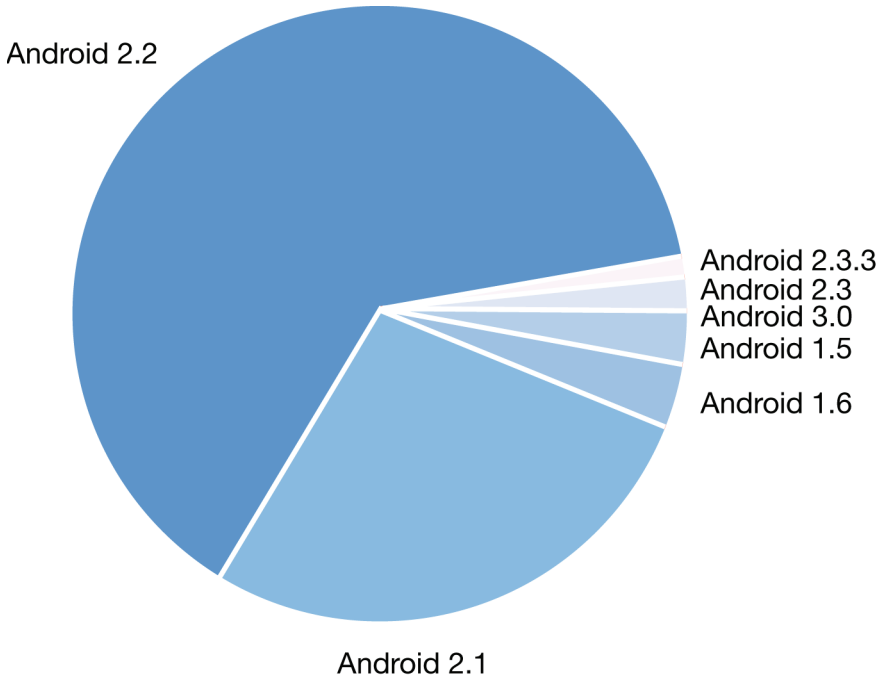
As a disclaimer, I am a designer. And while my knowledge of programming has increased a good amount while creating this document, it should be known that this document is written from the point of view of a visual designer. Also, although I have spent a good amount of time on research, this document should be considered a designer’s translation of Android and therefore subject to slight error and bias.

Know your market

Worldwide, there are over 90 Android devices that run the gamut of the operating system versions. While there are still a good amount of devices running Android 1.6 and below, this document is specifically directed toward Android operating systems 2.1 and greater.

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.



This chart displays the current distribution of operating systems as collected at the beginning of April 2011.

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. During the ill-fated port that I mentioned earlier, while the assets looked fine on most devices, on a few they appeared dithered and blurry. The fact that you are designing for multiple resolutions should be in the front of your mind through the entire design.

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%

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. 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 pixel resolution that you should be designing. With this in mind, Android has developed four generalized resolutions and four generalized densities for thinking about device screens. It breaks down like this:

There are four generalized sizes.

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

And four generalized Resolutions

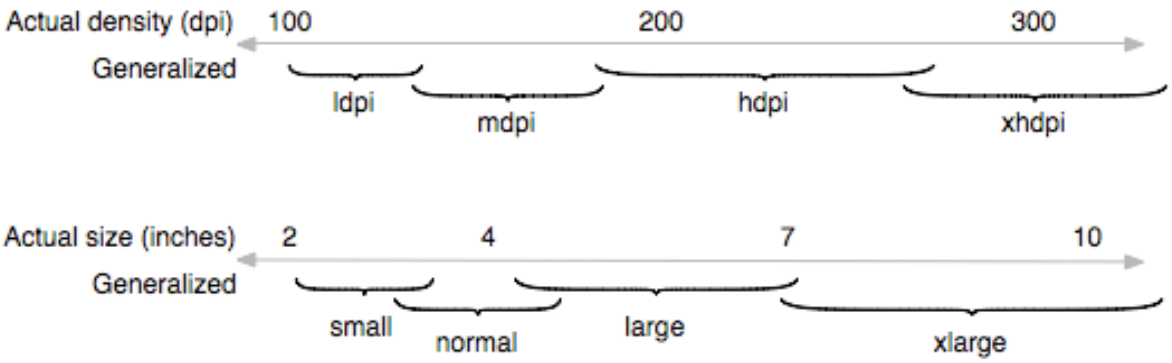
- ldpi (100-120 dpi)
- mdpi (120-160 dpi)
- hdpi (160-240 dpi)
- xhdpi (240-320 dpi)

How this affects layout

When creating wireframes for an Android app, it is wise to create your design in 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 this step will make for a better application.

How this affects design

It is good to keep things simple and clean with Android. Keep in mind that all assets need to scale to the different DPIs (dots per inch), however if you are intent on creating complex images, make sure you learn and follow Android's naming conventions (see "Naming Conventions".)



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

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

	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen		
Normal screen	WVGA800 (480x800) WVGA854 (480x854)	
Large screen		
Extra Large screen		

For the most part, screen sizes and densities correlate.

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

Being open source, Android is extremely flexible about UI elements and asset standards. As long as it technically can work, it works. However, there are patterns that are important to note. I took a myriad of screenshots and measured the various navigation elements and after some 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 consistent. Note, these are for operating systems up to 2.3 (Gingerbread). Honeycomb is a special case. Since currently the only available Honeycomb devices are mdpi, the assets are handled differently.

The tab bar

There's little that differs between iPhone and Android's tab bar and not much that's notable in its design. We work with tab bars every day in mobile, and it serve the same purpose with Android as it does with the iPhone. The only real difference is that Android, they can be top or bottom aligned.

The standard tab bar.

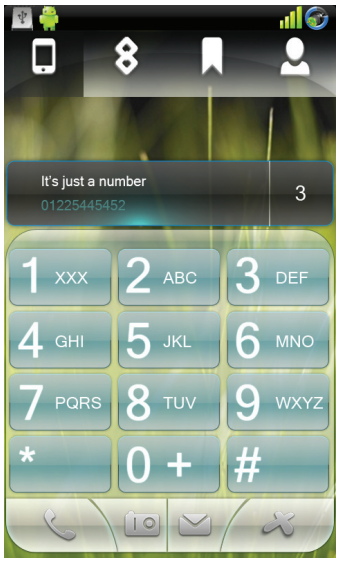
There is a somewhat standard Android tab bar, but it is somewhat unsightly. The prettier apps generally opt to make their own.

The consistent size

In all of my searches, I have yet to find a pixel size for the tab bar. Neither standard, regular, or suggested. So after measuring a sampling of screens, I have invented my own starndard: "consistent".

Consistent Android Tab sizes

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

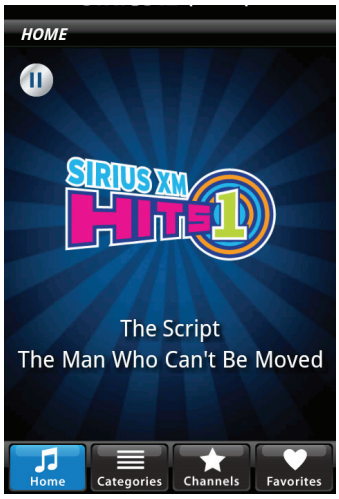


Slick and transparent

It came from Cupertino

Perhaps this is too editorial for this document, but Android applications that try to look like iPhone applications don't work 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? Using iPhone-style navigation is likely not intuitive for seasoned Android users.



Not so much

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 nav bar mixed with what you would find in an action sheet or toolbar. 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

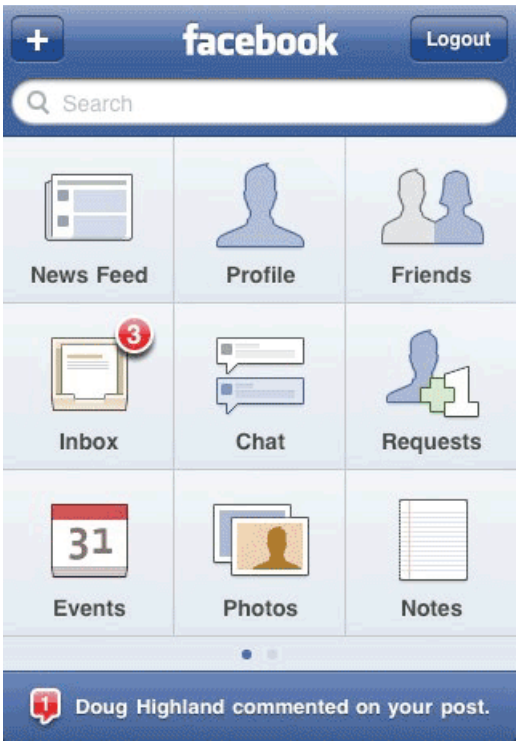
The importance of being an options menu

The importance of the options menu cannot be understated. The 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.

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, they use the options menu.

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

vs.



The context menu

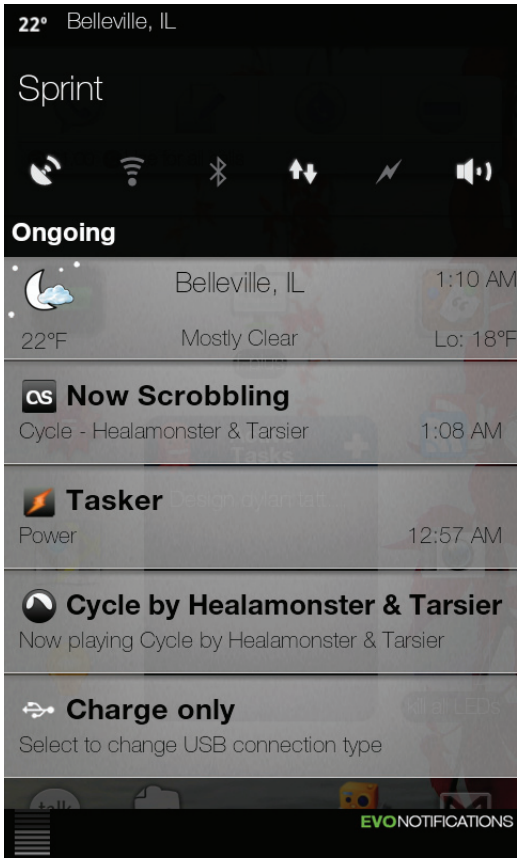
The context menu is similar to the right click on the desktop. The user will touch and hold to bring up the menu which will provide commands that pertain to the selected activity.

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



Sleek, pretty, hidden

When to use an option menu vs. a context menu

Selecting the right kind of menu can be a bit confusing. To sum it up, any activity that is global would either go in the options menu or a list view. Activities that pertain solely to the content of a cell in the list view would summon a context menu. Option menus

should 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- such as with your contact book, where you need to have a number of options attributed to each field of the list view - the context menu is unavoidable. 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 that it is there. 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. The context menu should be treated as advanced user commands - things that can be done another way but are faster if you know that the menu is there.

Short names in the option menu

Much like the iPhone Springboard, the Android Options Menu will truncate long names, so keep it short.

Paper beats rock, dialog 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 can continue.

Dim and hide

There are times when an item that is in the option or contextual menu does 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 (grey) it out. If you have an option like this in the contextual menu, hide it altogether.

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.

In Android, a dialog box is used for 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.

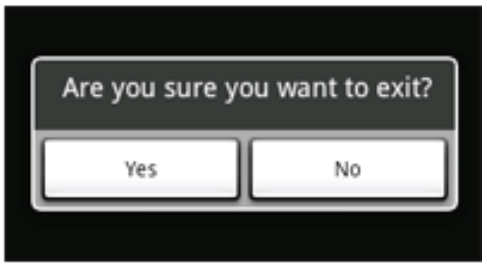
The category of dialog boxes contains a variety of different types of modals.

A Custom Color Picker

Alert dialog

Similar to the alert on the iPhone, the alert dialog can take a few different forms. An alert dialogue is used to display a warning, a text message or a choice (such as quitting an application).

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

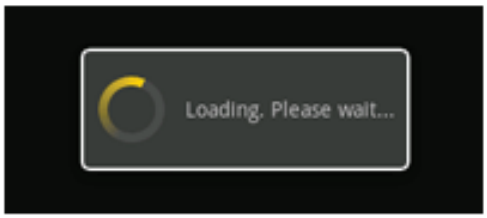


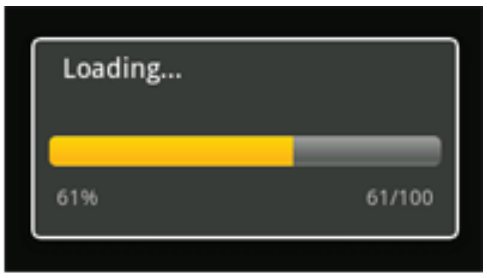
Examples of Alerts

Progress alert

The progress alert comes in two forms:

- The spinning wheel represents an undefined progress and 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.





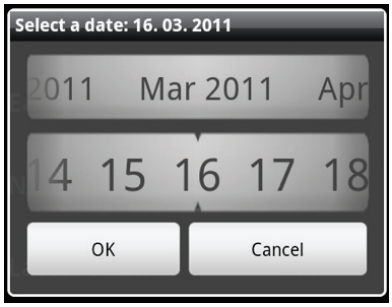
Android's fantastic what-you-see-is-what-you-get date and time picker

Android's date and time picker is not much to look at. Certainly it is sparse, utilitarian and functional.

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.

At bottom right is an example of a custom date picker. Not fantastic, however it is an 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 #ff191919). 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 to 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.

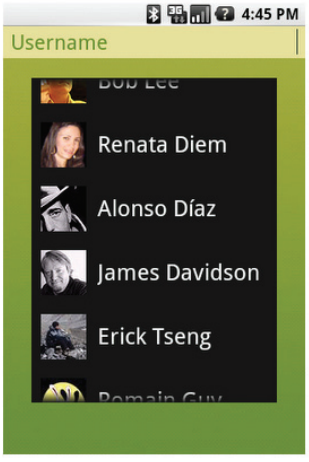
Unfortunately, this has terrible results when the user swipes through a list on a custom background.

To avoid this 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. Later in the document, we will discuss Android standard bounding sizes, but first 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 draw-9patched without losing their finesse.

For example:

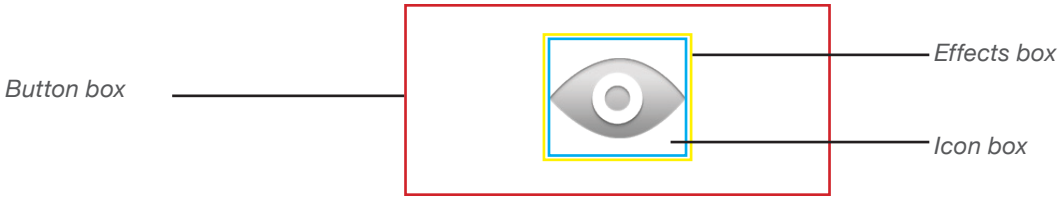
Your Icon

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 pixels wide and 75 high. As indicate by the red box.
- 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 *(ldpi)*	Medium density screen *(mdpi)*	High density screen *(hdpi)*
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 (preferred, width may vary)	16w x 25h px (preferred, width may vary)	24w x 38h px (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 iPhone, an 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 launcher 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.

Standard Android icons

The standards and styles

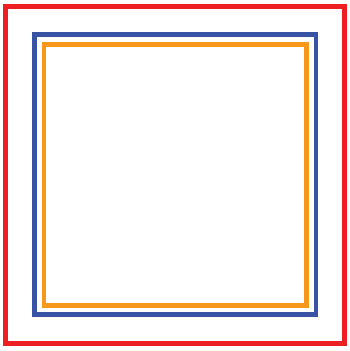
When it comes to designing the launcher, Android has a surprisingly high number of rules. While no one is going to reject you if you break these rules, this is what Android expects out of a launch icon.

According to 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

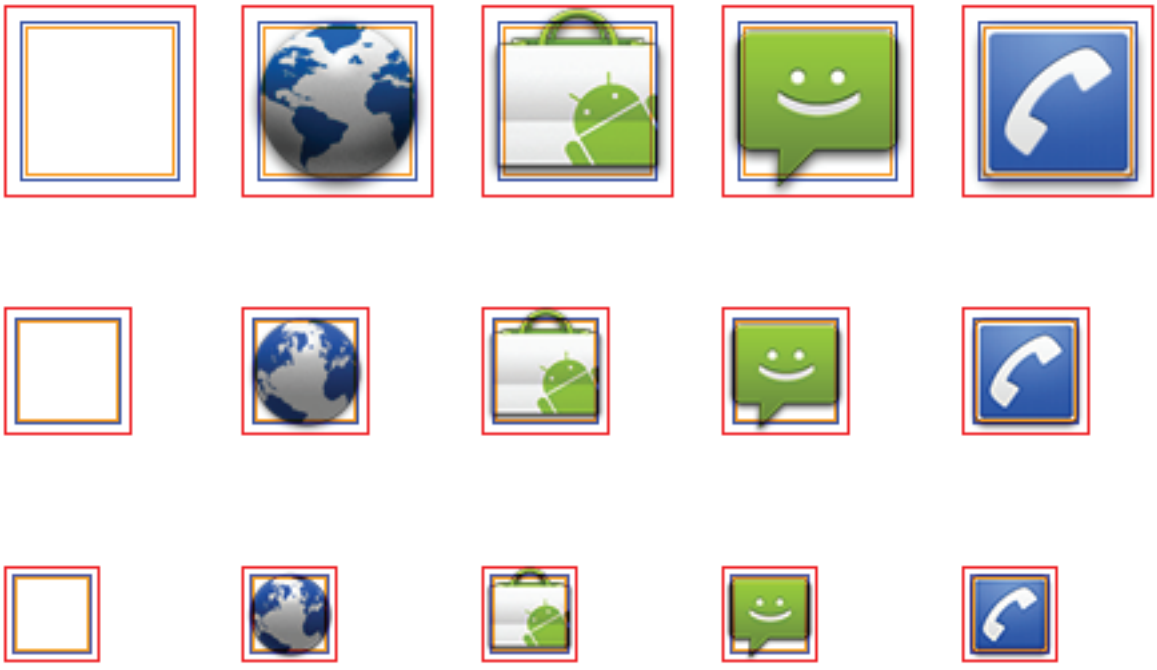
The reason that Android icons can be practically any shape has to do with the fact that their icon falls on a grid. To make an icon correctly, these rules must be followed. This applies to every icon you make. The figure on the following page displays how an Android icon is set up.



Learn to love this box

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.



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 (hdpi) screens:

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

Launcher icon dimensions for medium-density (mdpi) 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 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

The launcher icon drop shadow effect

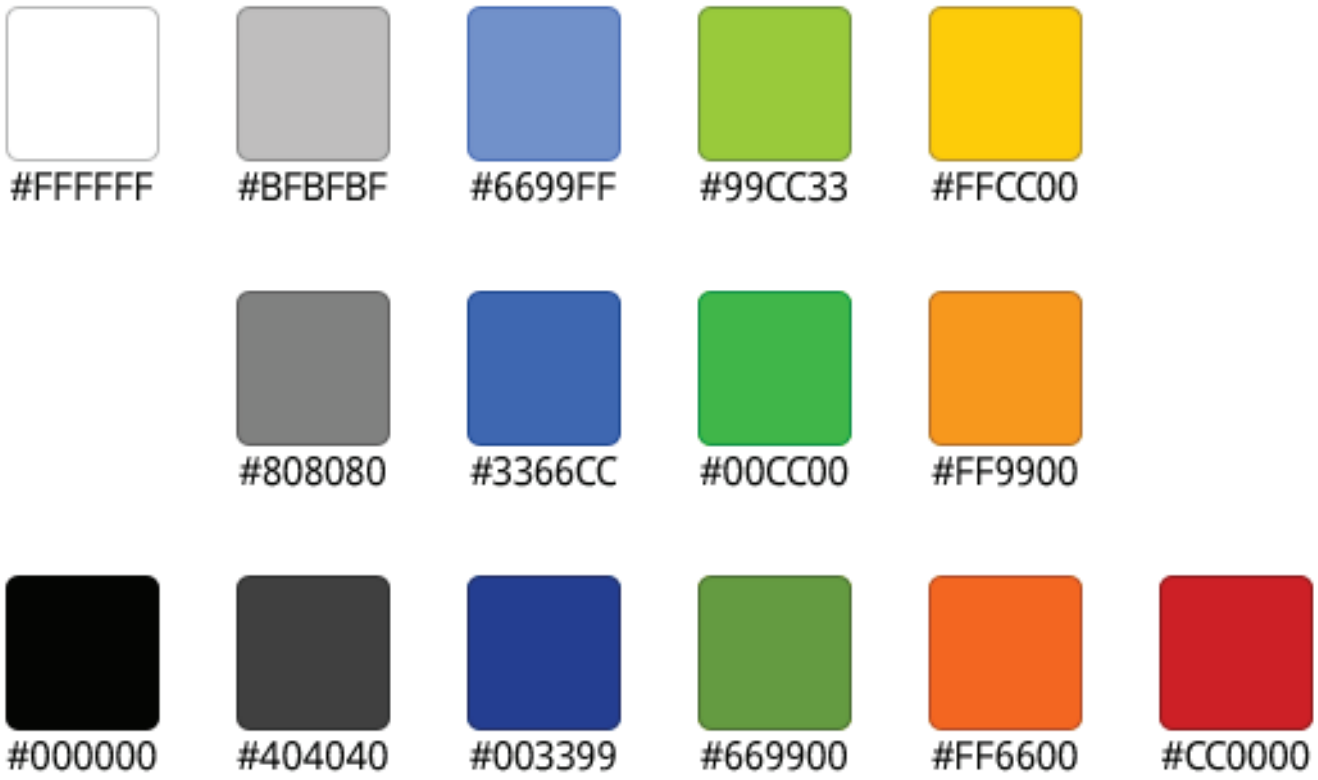
In order to keep your icon consistent with the others on the spring board, a very specific drop shadow should be used. Below are the drop shadow amounts for Photoshop and Illustrator at all screen resolutions.

Photoshop

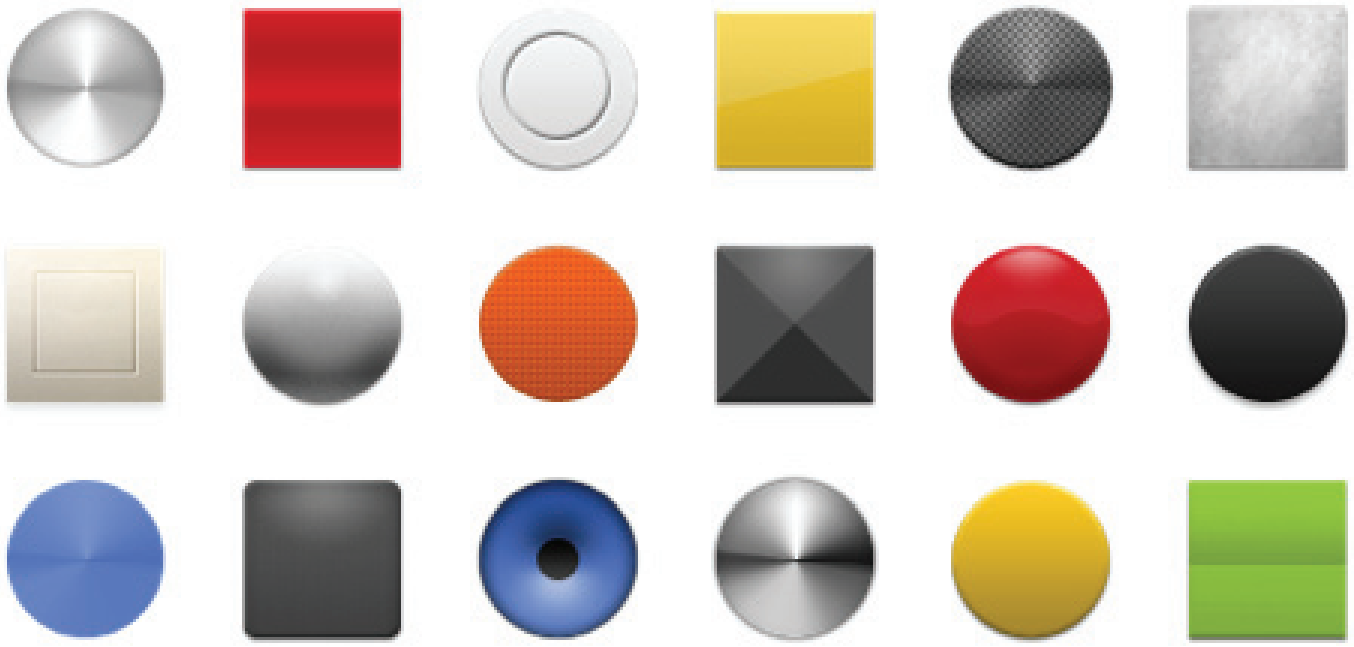
- HDPI: #000000 75% opacity, Distance=1.5 Size=4.5px, Angle=90
- MDPI: #000000 75% opacity, Distance=1 Size=3px, Angle=90
- LDPI: #000000 75% opacity, Distance=.75 Size=2.25px, Angle=90

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



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 (hdpi) 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

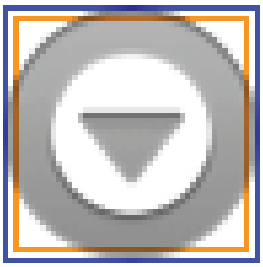
Menu icon dimensions for low-density (ldpi) screens:

- 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 2px
- size 2px

Inner bevel:

- depth 1%
- direction down
- size 0px
- 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 the 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 2px; size 2px
- 3 - Outer glow: white | 55% opacity; spread 10% | size 3px
- 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 bar 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 left transparent in the png.
- Outer Glow. #000000, 25% opacity 3 px.

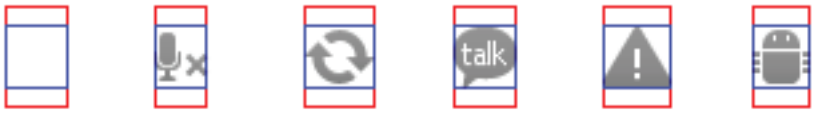
Status bar icons

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:

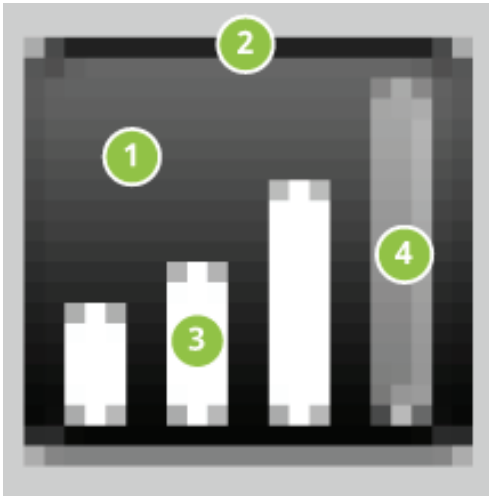
- Inner content should be subtracted and left transparent in the png

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 0px | 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 box is completely customizable. Here are some guidelines for building dialog icons. Dialog and list view icons are pretty much the same except that the Android's convention is to give the list view 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 (hdpi) screens:

- Icon: 48 x 48 px

Dialog icon dimensions for medium-density (mdpi) screens:

- Icon: 32 x 32 px

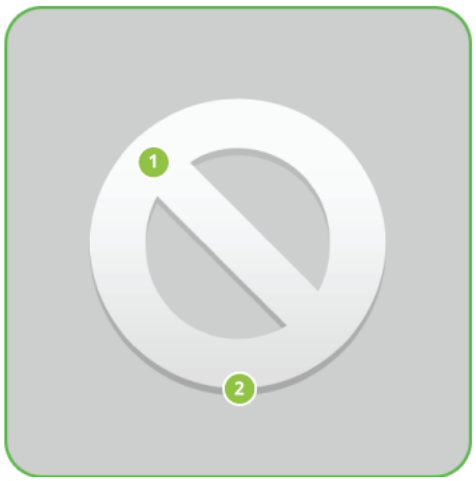
Dialog icon dimensions for low-density (ldpi) screens:

- Icon: 24 x 24 px

Dialog and listview effects

These are the effects for an icon made for a standard Android alert. The 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 1px
- size 0px

Listview icon

1 - Inner shadow:

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

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 a mini application extension that runs on the home screen of the Android. The widget displays the application’s most relevant information at a quick glance. Users pick the widgets they want to display on their Home screens by touching and holding an empty area of their Home screen, selecting Widgets from the menu, and then selecting the widget they want.

The widget has three main pieces - 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 may be 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 multitask 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.

Everyone loves telling time

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.

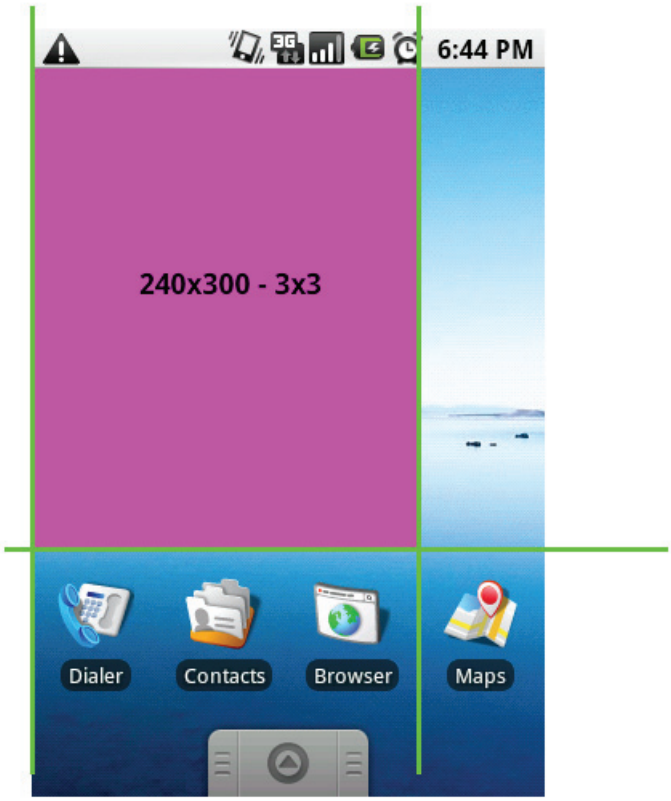
If your widget has any toggle functionality, (such as a music player), make sure that the buttons have three states: inactive, pressed and active.

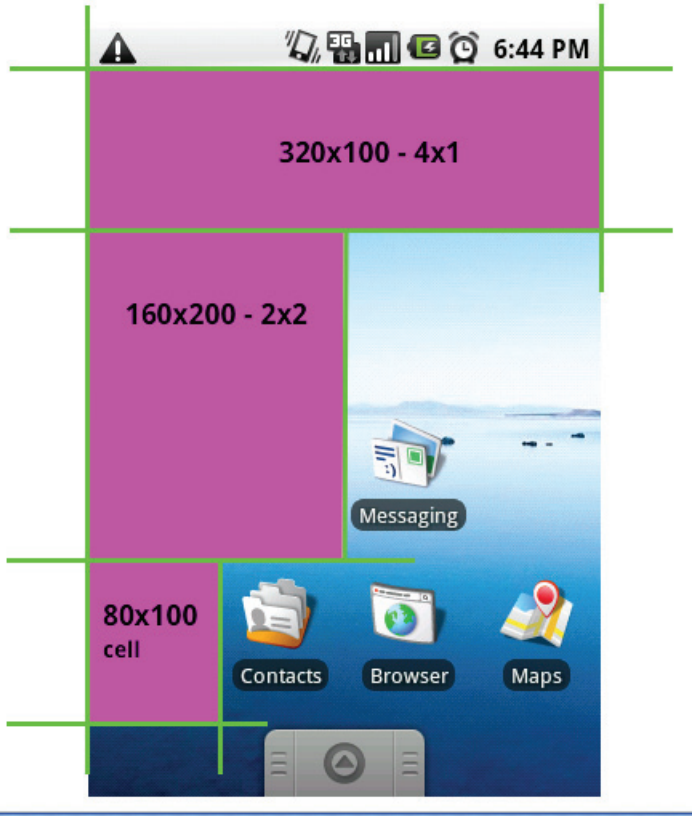
Widget sizes

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.

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

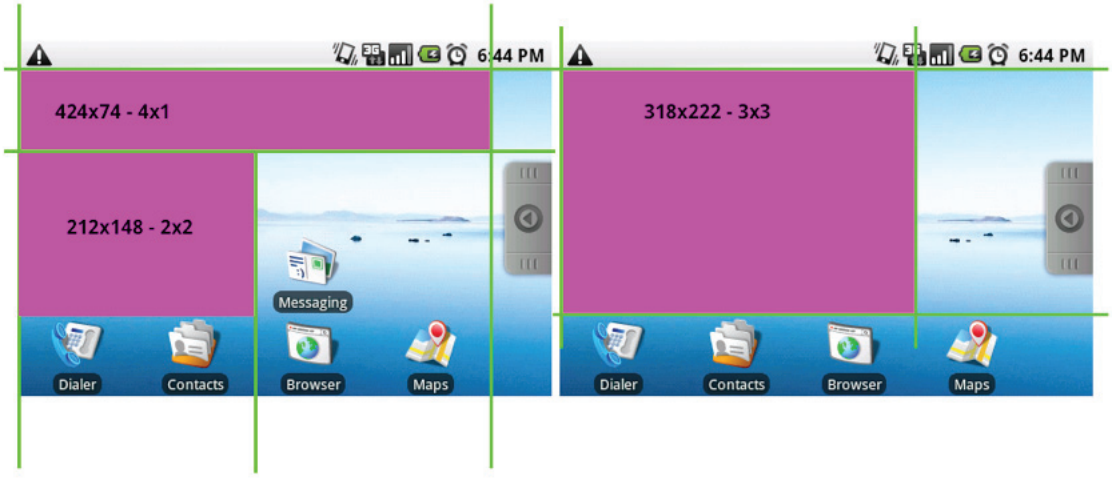
- Cells Pixels
- 4 x 1 320 x 100
- 3 x 3 240 x 300
- 2 x 2 160 x 200



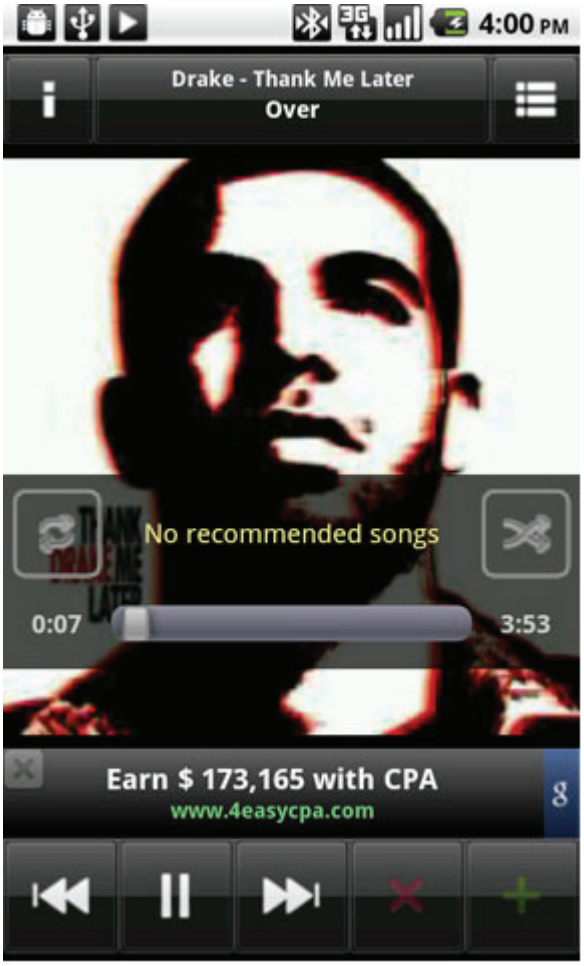


In Landscape mode, the each cell is 106 x 74. The standard widget sizes are:

- 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 Music widget, for example, is a popular dynamic music playing widget that actually takes up 4 x 4, in other words the whole screen.



Full Screen Widget

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 users want from a given app and how we present it is constantly changing.

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 an image that will be allowed to stretch and expand, leaving the rest of the image intact. You should create draw-9patched assets if they are solid colored, such as a button or tab, or if they have transparency, such as an icon. You should never attempt to draw9patch a complex image, such as one that contains multiple effects or one that has a rich gradient, as the complexity of the image will certainly be compromised when you start stretching pieces.

An asset that shoud be draw9patched

An asset that should not be draw9patched

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 used by any designer taking on an Android project. Draw9patching takes some experimentation.

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

Figure 1

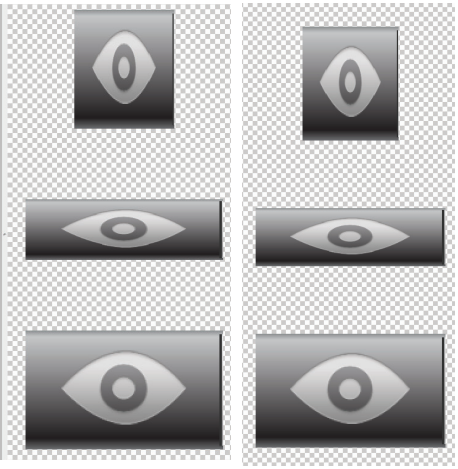


Figure 2

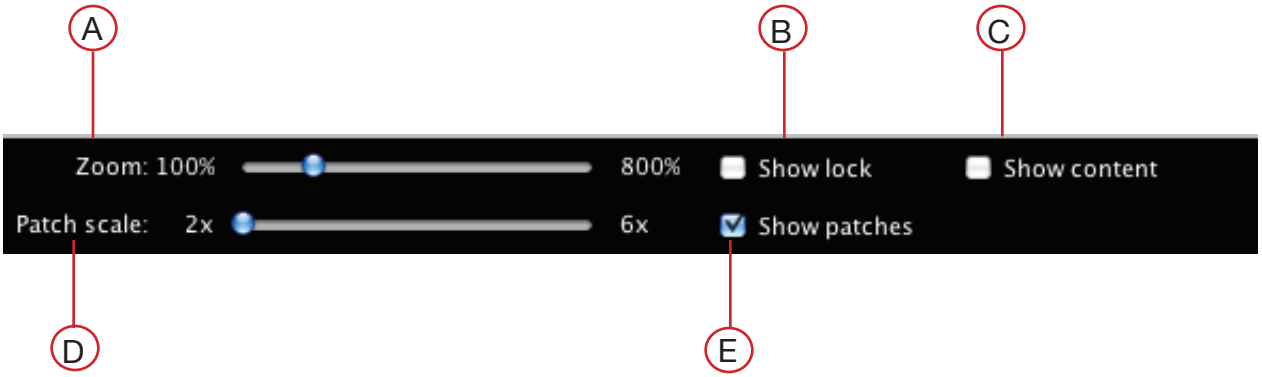
The right pane of Figure 1 displays what an asset looks like when it is draw9patched and scaled. In Figure 2, the png has not been scaled yet. If the asset is used without draw9patching, this is how the image will appear on different devices.

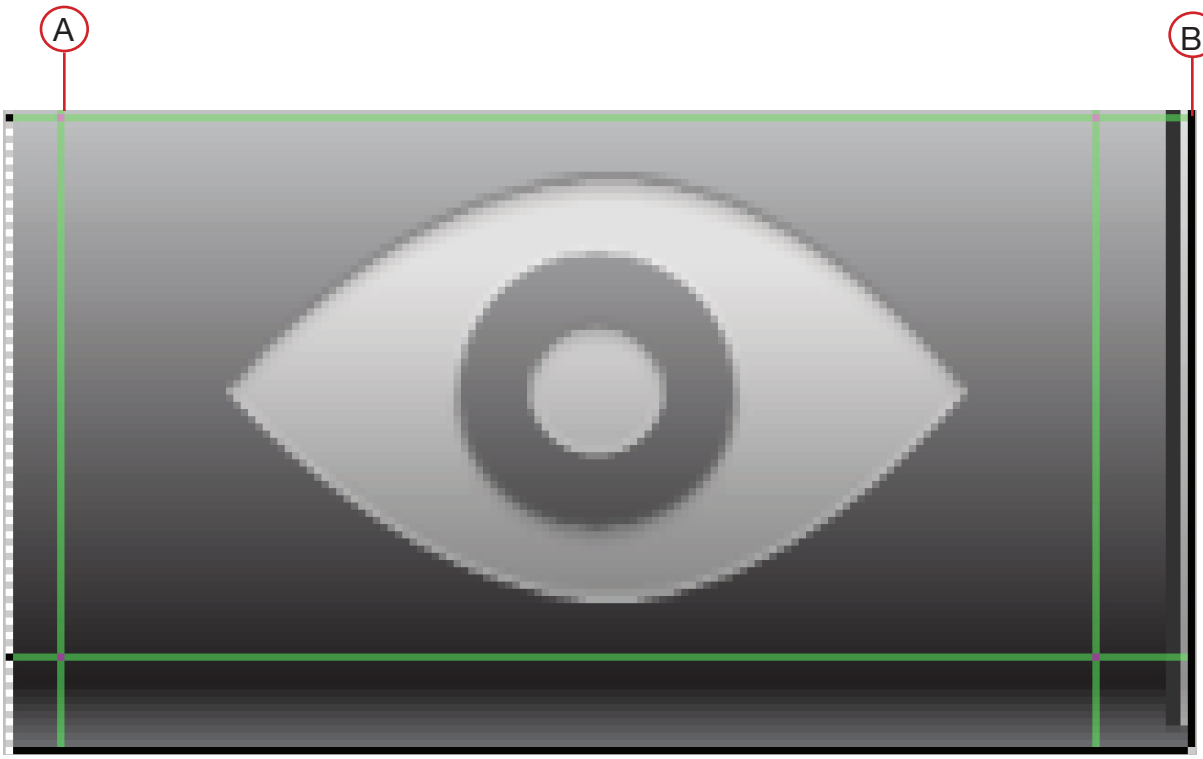
These are the very simple tools to the draw9patch program:

- A - The slider will zoom into your main work space.
- B - Show lock will display the non-drawable 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

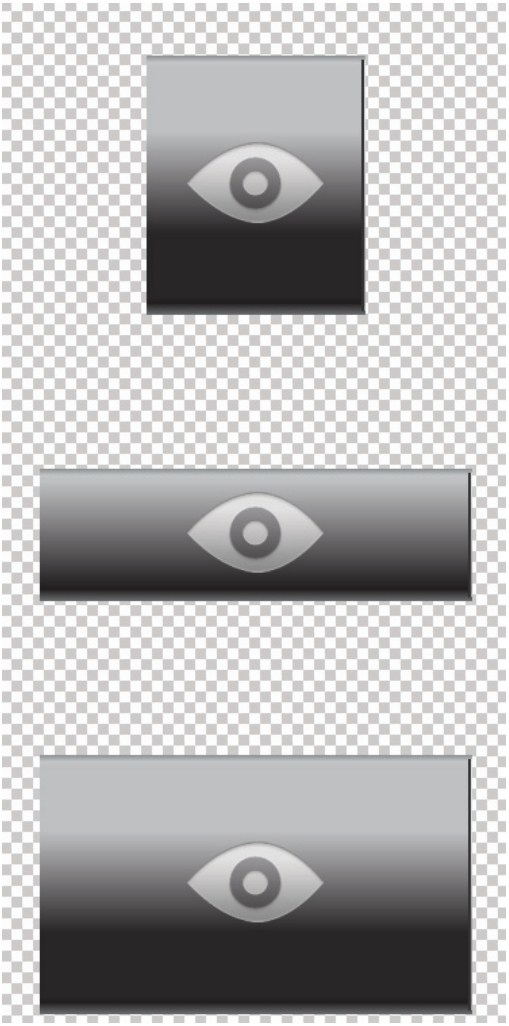
- B - Click and drag your mouse on the right and bottom side to select the area of the image that will be affected by the draw9patching.

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.

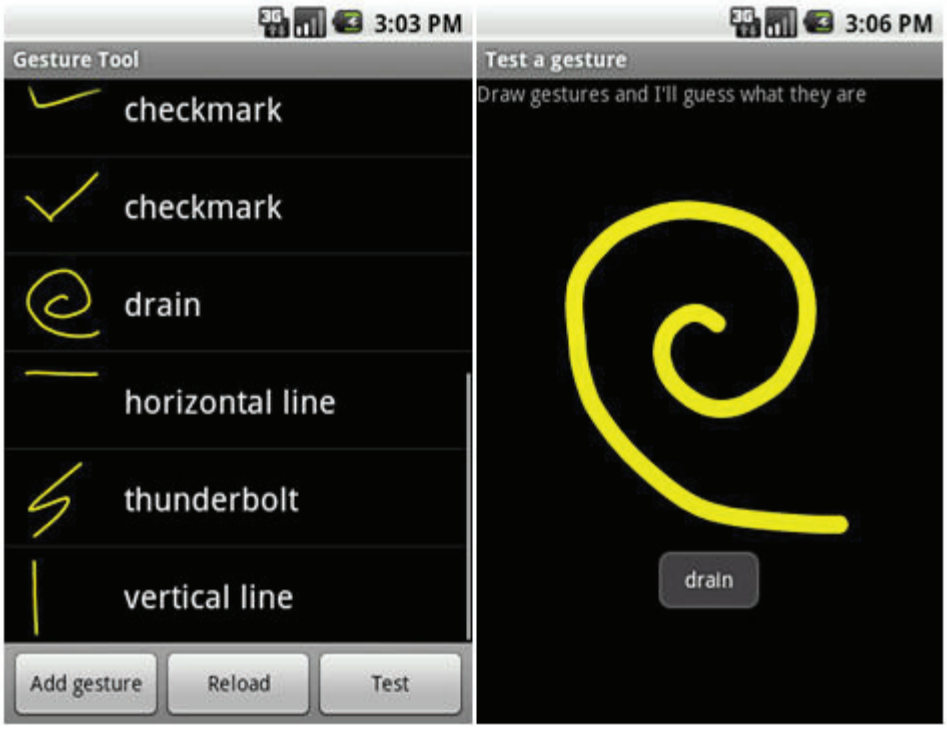


gestures

Android Gestures can be a bit tricky to understand. While not necessarily advisable, the types of gestures that can be created are infinite. To understand gestures, you must break them down into two distinct classes; motion events and gestures.

Motion events

A motion event is the actual touch event on the screen. It tracks x,y coordinates and pressure and is defined by a touch up or touch down. An example motion event would be “drag and



Every app deserves a thunderbolt gesture

drop”. Selecting the object to drag and drop would be the touch down on the object (or more specifically, the object’s coordinates. The touch up would be the “drop”, using the new coordinates as a destination for the object when the finger is lifted.

Gestures

Gestures define the series of motion events that create a solid movement. To continue with the drag and drop analogy, the gesture would be the movement of the object.

How this affects the UX

The pressure of a tap is decided by a motion event and the movement is decided by the gesture. A gesture should be considered when working with UI elements. For example, if a screen is UI heavy, it is advisable to keep swipes, for instance, to a minimum.

Just because any action can be a gesture doesn't mean it should be

With the Gesture Builder app in the Android SDK, you can turn almost any motion into a usable gesture, which is pretty neat to think about - Until you consider how that will effect user experience. If you are contemplating adding an unusual gesture to your application, make sure you have a good reason for doing so. Custom gestures can be pretty great for game design, but not so much for utility apps.

Standard gestures

Even though Android's gestures are limitless, there are still some common gestures that are consistently used. It is good practice to stick to these gestures as they are what Android user's are used to.

Gesture	Action
Tap	Selects an object, navigates through different screens
Flick	Scrolls up and down
Long Tap (Tap and Hold)	Brings up additional options, such as editing and contextual menus
Double Tap	Zooms in
Pinch Open	Zooms in
Pinch Close	Zooms out

gingerbread

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

Application management

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

Updated UI

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.

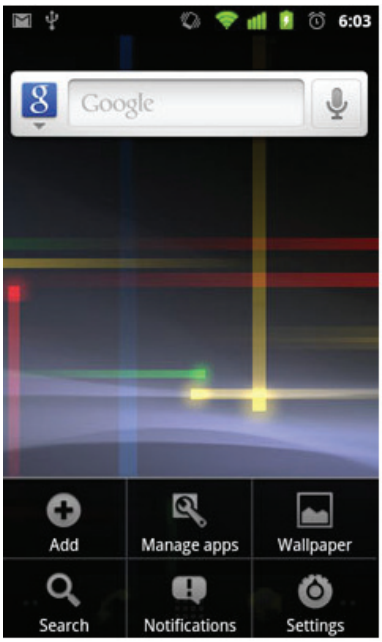
New Gadgetry.

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

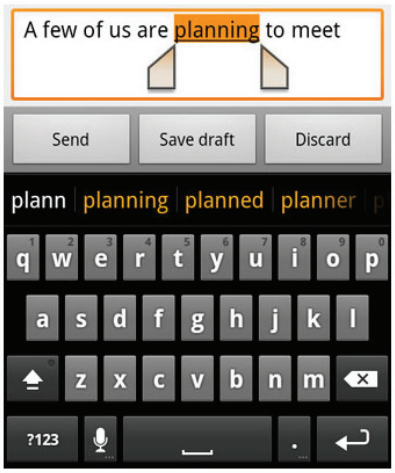
This includes:

- Gingerbread will offer support of near field communication. Soon users will be able to tap their devices against Near Field Communication 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

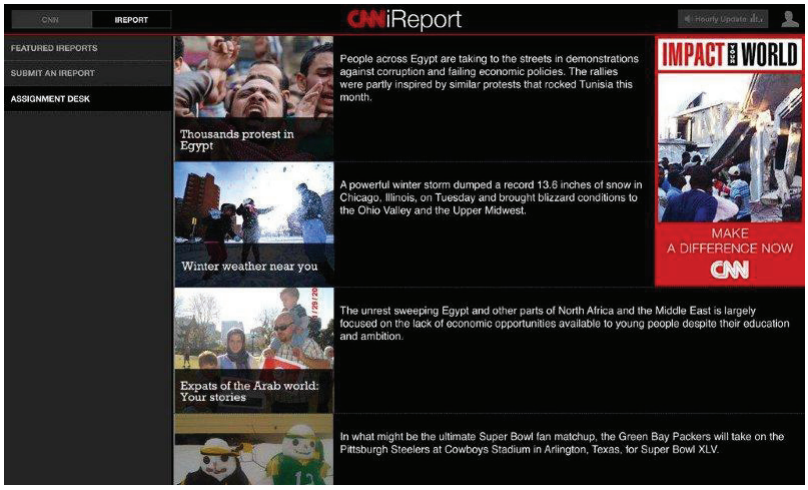


Android 2.3

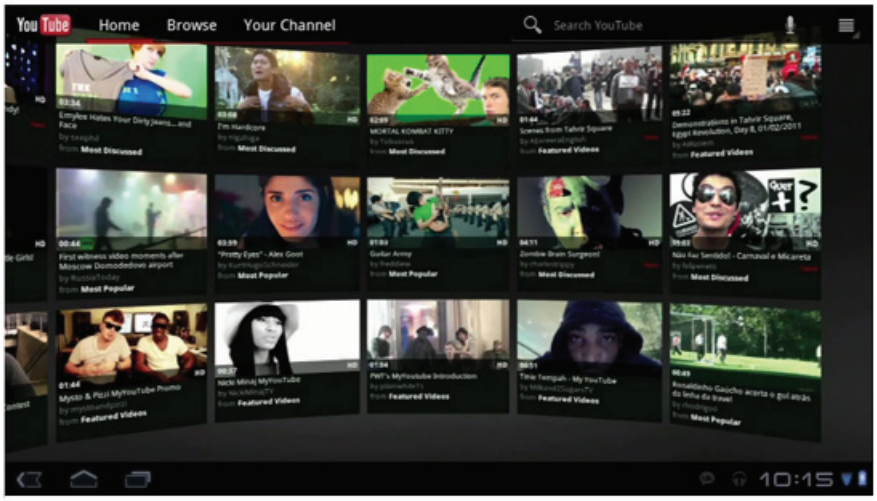


honeycomb

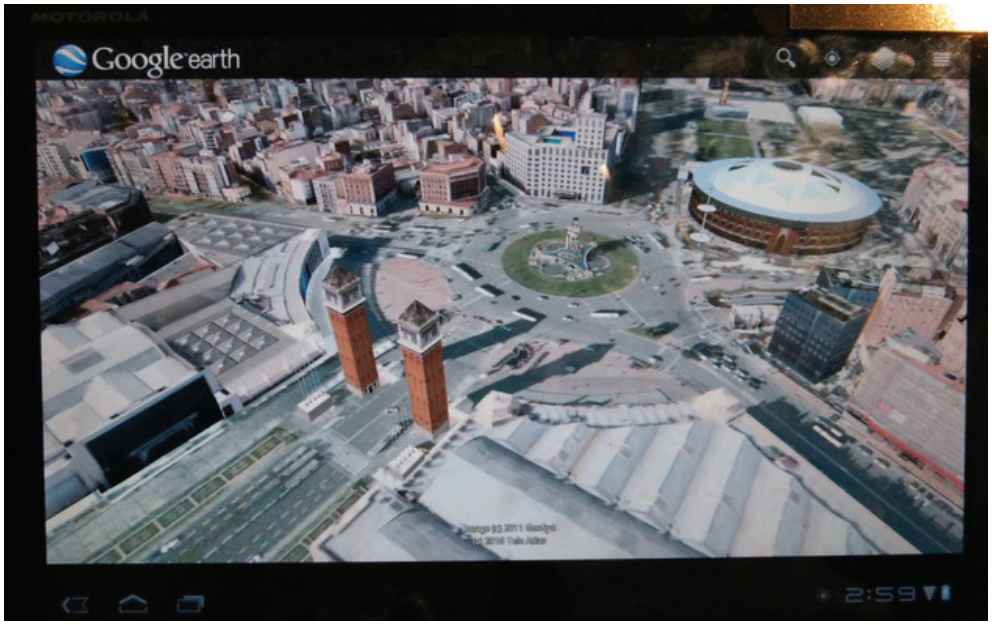
Honeycomb is a game changer for Android. By introducing an optimized UI, holographic themes, intuitive multitasking and a redesigned widget structure, Honeycomb has created a user experience that not only makes sense functionality-wise but also has the capability of being stunning visually. While there are not many Honeycomb optimized applications on the market yet, the ones that have been developed are well constructed and beautiful. CNN, YouTube, and Google Earth are three examples of remarkable Honeycomb applications.



The CNN Application



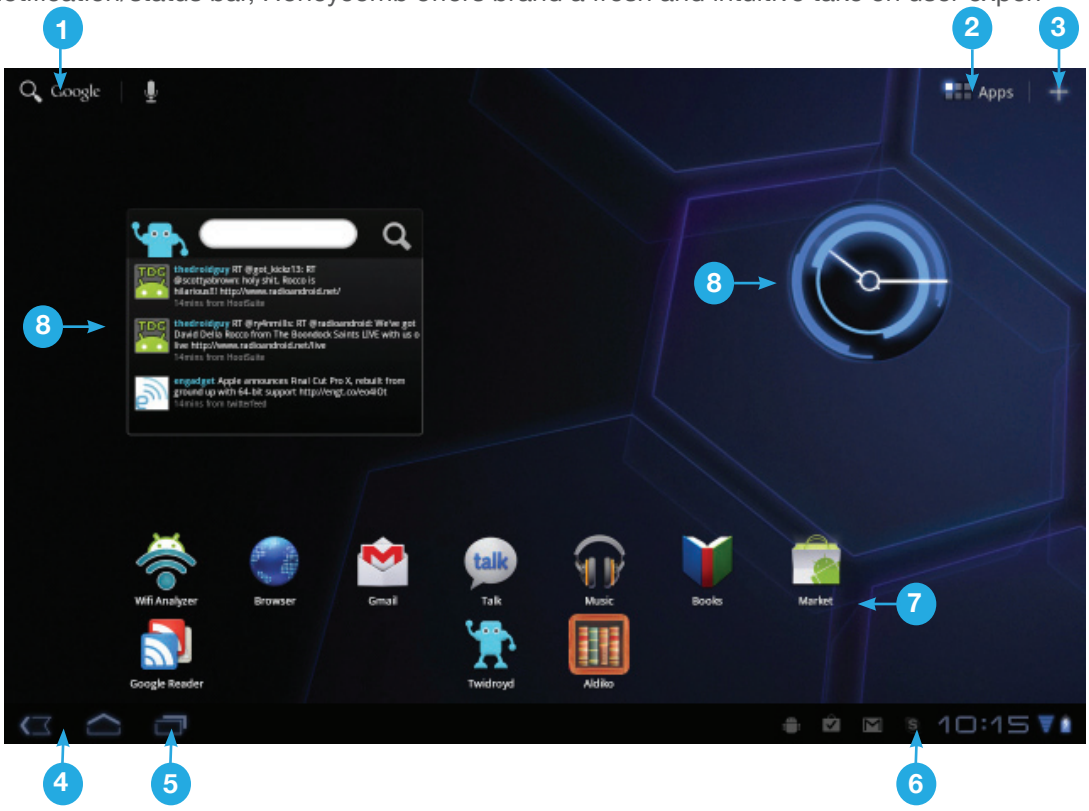
Youtube Application



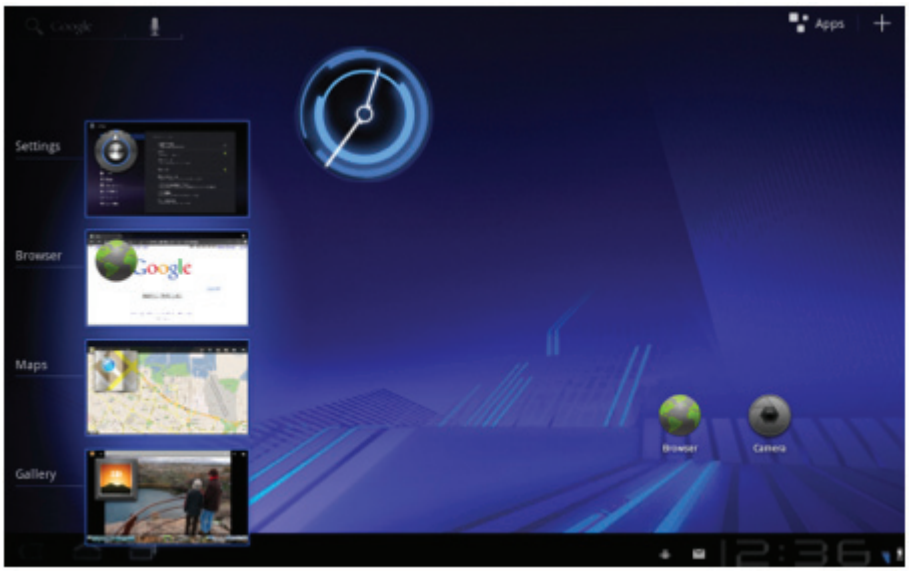
Google Earth

Dissecting Honeycomb

Honeycomb introduces a brand new UI structure to Android. With the advent of the action bar and notification/status bar, Honeycomb offers brand a fresh and intuitive take on user experience.



- 1. The top of the screen in Honeycomb is reserved for the action bar. The action bar serves as both tab bar and menu, including both buckets of functionality and the means to edit the fields within the screen. On the home screen, the left side is occupied by an editable Google search field controlled by both text input and voice.
- 2. This icon on the right side of the action bar will bring up a complete list of apps stored on the device, as well as the Android app store.
- 3. This plus icon will bring up to a customize screen, allowing the user to select widgets, wallpapers and bookmarks to store on the home screen.
- 4. This is the System Bar. While on older devices it is placed at the top of the screen, on Honeycomb it is placed at the bottom. On mobile devices the back and home button are hardware elements; in Honeycomb they are manipulated by these icons. The system bar can be placed in a “lights out” mode dimming it completely, useful for activities such as watching movies.
- 5. This icon stores all active applications, allowing the user to easily switch between apps, optimizing multitasking in a unique way. A screenshot is saved of the last view of an application is displayed when this icon is tapped.



App History, supreme for multasking

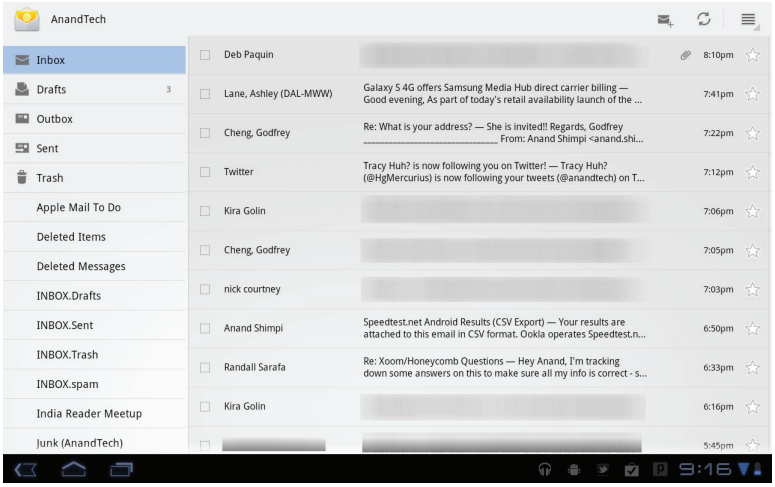
- 6. This portion of the status bar is called the notification bar. It replaces most other notification, and it alerts the user when they have new messages, Facebook updates, etc.
- 7. These are app shortcuts. The rest of the apps are located in point 2. While by default they are organized at the bottom of the screen, with Honeycomb, the user can drag and drop the icons wherever they please.
- 8. These are widgets. When long tapped, the top right corner becomes a trashcan. At this point they can be drag and dropped to be moved or removed.

New UI elements

Honeycomb introduces a few key new UI elements. At the moment the few Honeycomb devices that have become public run at mdpi with an extra large screen.

Fragments

In order to get the most out of the screen size, fragments should be utilized. Fragments are separate portions of an application that function independently but work seamlessly together. In the Gmail app, the left-side fragment displays all of the user's different mailboxes. When the



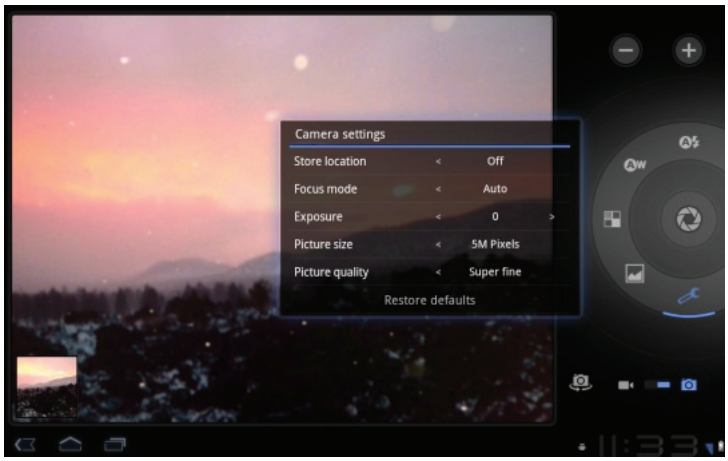
Examining fragments

user taps on an email, the left-side fragment is replaced with the list of emails seen on the large right-side panel which is then populated by the body of the email messages.

Fragments can be any size, and can be placed on the left or right side of the screen. There can

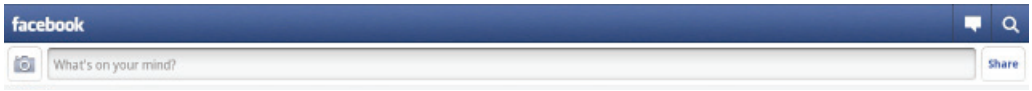
also be multiple fragments on one screen. It is truly remarkable what can be done with the multiple panes of fragments. Let's look at the CNN app for example.

The left pane controls the content of the larger right pane. However, the user can swipe the main pane to the right which in turn effects the left pane. The new fragment API has been made compatible with operating systems starting at 1.6 so that applica-



The camera application has a very interesting fragment

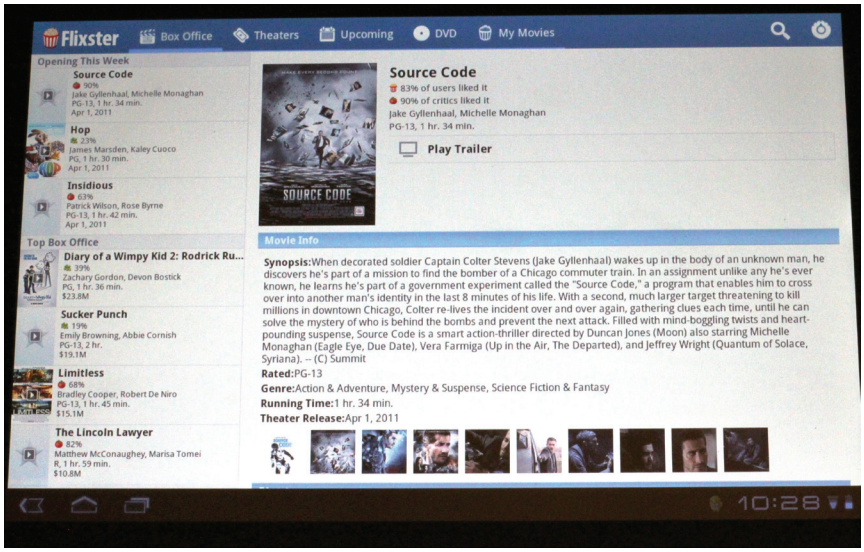
tions compatible with older operating systems can use fragments to make tablet-compatible interfaces



The action bar

The action bar

The action bar is at the top of the screen and replaces the title bar. By default, the app logo is positioned on the left-hand side, however there are apps (such as Pulse) that opt to do otherwise. They contain both tab bar items and tool bar items.



The Flickster app uses tab navigation in the action Bar

The Movies app uses the Action Bar to organize different buckets of functionality, much like a tab bar. From here you can access:

- Box Office
- Theaters

- Upcoming Movies
- DVDs
- “My Movies”

The Mail app uses the action bar like a tool bar, allowing the user to:

- Switch between accounts
- Compose mail
- Search through messages
- Refresh messages

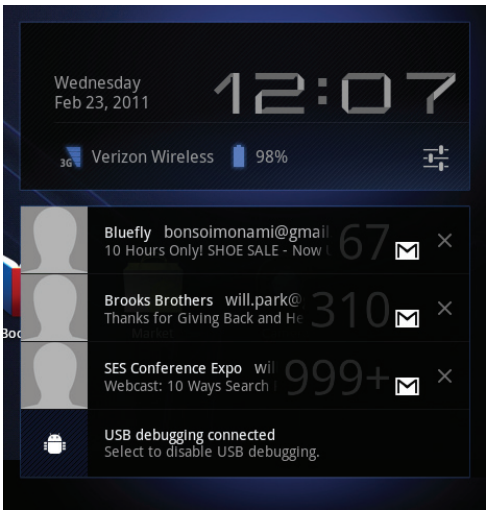
The far right of the action bar will almost always contain a preference or settings button that triggers a drop down menu.

The action bar’s dimensions are 1280x48 mdpi. It can be customized and while the selection colors are programmatic, the color can be altered in code.

System Bar

System bar

The Honeycomb system bar contains navigation previously controlled by hardware back and home buttons. Additionally, the stacked picture icon will allow you to access recently opened applications. This is not a customizable element, however icons can be made for the notification bar. It is 1280x48 px.



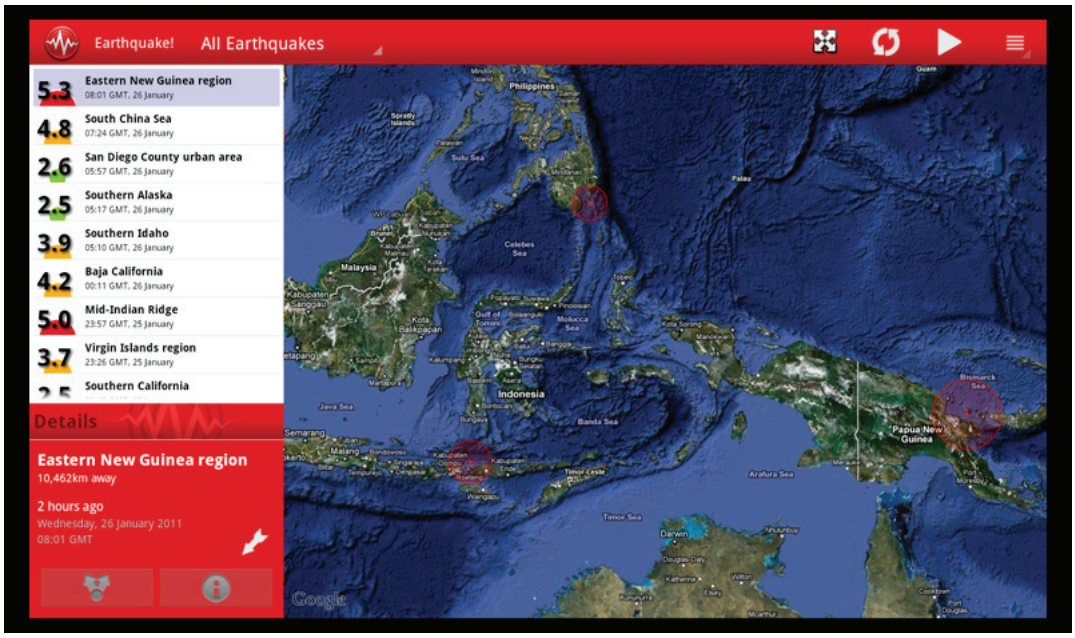
Notification popup

The right side of the system bar contains notifications. When those icons are tapped a small popup dialog indicating the notification expands. These dialogs can contain pictures (such as a contact’s picture). If the clock is tapped on, a larger popup opens containing a larger clock read out and the list of notifications. From this popup, the user can access their device settings.

Honeycomb options menu

Once and a while, an application will have use for an options menu. When this occurs, an extra icon will be present on the left-side of the system bar. Most of the functionality of the options menu has been replaced by the absolute diversity of the action bar. It’s a holdover from previ-

Option menu Icon



Typical listfield in a fragment Menu

ous operating systems as there aren't many optimized Honeycomb apps on the market. The Honeycomb option menu works the same as did for previous operating systems, sliding out from the center of the screen above the systems bar.

Listviews

Listviews usually show up in fragments. In Honeycomb, they can be customized with a background and custom separations.

Stacking widgets

Brand new widgets.

Widgets have always been one of the coolest components to the Android. Now a Widget can deploy a certain type of listview, called "stack of cards."

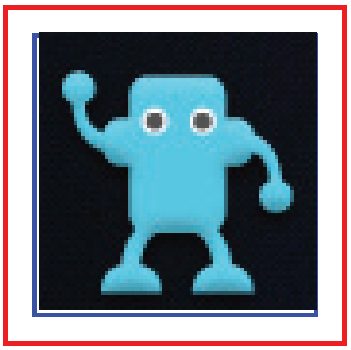
The Youtube widget, for instance, uses the "stack of cards" to allow users to flip through an assortment of videos without going into the Youtube application.

Widgets are handled the same as they previously have been. The screen is divided into 74 px cells. Your widget can be virtually any size, but must stay within the structure of the cells. Android suggests taking off two pixels for padding. So that is 72 px to work with, multiplied by how many cells you plan on taking up.

Honeycomb icons

Honeycomb presents new types of icons to create: The launcher icon, action bar icon and the notification bar icon. I must stress that I have attempted to come up with the proper bounding box approach to these icons, but until they are officially released, the sizes presented should be considered approximate and unofficial.

Launcher icon



The launcher icon should follow the same rules as it does for mobile devices. It should be vibrant and colorful. Even though Honeycomb devices are only mdpi at the moment, the icon should be made with the same pixel dimensions of a ldpi mobile device icon.

Launcher icon dimensions for medium-density (mdpi) tablets:

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

This will create a problem unless you add a qualifier to the png's name. The asset should be in a folder called "project/res/drawable-xlarge-mdpi" or better yet "project/res/drawable-api11-mdpi" (see naming conventions)

Action bar icon

Action bar icons are tricky. After measuring a couple different apps. I was unable to find a size that was totally consistent. I believe that best practices would be this:

Action bar icon dimensions for medium-density (mdpi) tablets:

- Full Asset: 48x48
- Icon 20X20

That probably seems like an awfully big bounding box, however, Action Bar Icons look best when they are deeply nested within a button. They should be a simple color with a subtle gradient and subtle effects.

Notification bar icons

Notification bar icons appear when an application has something to tell you, such as a new email or weather alert. I am unsure of an appropriate bounding box size, for the time being I will only give you a best practices size of the asset.

Notification bar icon dimensions for medium-density (mdpi) tablets:

- Icon: 24x20

Most Notification Bar Icons are grey R: 84 G: 84 B: 84, though they can be any color. They should be simple, minimal and without effects. The inner content should be subtracted and left transparent in the png.

Other Honeycomb features

Multi select

With Honeycomb, the user can select more than one item in a list by long tapping on the field. This will allow for easier management and editing.

Optimized web browser

The browser inside Honeycomb has tabbed web pages, autofill and bookmarking capabilities.

Easier Copy and Paste clipboard

By long tapping on text, the user will be able to select text to copy and the option to copy that text or send it via email, bluetooth, Facebook, etc.

Cut and paste

naming conventions

More than any other mobile device, naming conventions are key in designing a quality Android app. As designers, we follow the rules of three DPIs. At least in my experience in working with Android apps, I have been asked to deliver assets in folders labeled HDPI, MDPI and LDPI. But in order to design a quality Android app, it is important to know how much further than this it goes.

Folder labels

Android assets are recognized by specific folder names that describe their contents. Understanding this will save time for development. If I were creating a Honeycomb app, for instance, a typical folder structure would read something like this:

Project Name/res/drawable-xlarge-mdpi

What does this mean exactly?

- **Res.** Res stands for application resources. From a design point of view, this means UI element assets.
- **Drawable.** A drawable resource pulls a singular visual resource or asset. This is the folder that your PNGs go into.
- **“xlarge”** This is the qualifier and is extremely important to note. In this instance xlarge describes the size of the device. The Xoom, for instance, is an extra large screen size that runs a mdpi resolution. If you only keep your assets in a resolution specific folder, the assets will not look right on the larger device. For example, the typical size of a MDPI launcher icons for mobile devices is 48x48 px. Even though a Xoom is MDPI, the launcher icon is 72x72. If you do not make the distinction of screen size, the asset that is pulled will be the wrong size and will look terrible.
- **“mdpi”** This describes the resolution of the app. MDPI, for example runs assets at 160 dpi, which is 1.5 times smaller than a hdpi asset. So in order to keep the asset looking correct, it needs to be copied and resized.

The following is a chart of qualifiers and how they will effect your assets:

Qualifier	Value	
MCC and MNC	**MCC - Mobile Country Code** **MNC - Mobile Network Code**	If you are tailoring an app to a specific country, you need to include the specific MCC. A list of codes can can be found at http://en.wikipedia.org/wiki/List_of_mobile country_codes The MNC is for tailoring the app to a specific network a list of networks can be found at http://en.wikipedia.org/wiki/Mobile_Network_Code
Language and Region	Examples: **en** **fr** **en-rUS**	If you are tailoring an app to a specific language, you need to add the language code. If it is regional it is comprimised of the language code and a region code. The region code is a 3 letter suffix starting with a r and ending in a country code. The list of country codes can be seen on pg
Screen Size	**small** **normal** **large** **xlarge**	This is your screen size. If you are designing an app to run on Honeycomb presently, there needs to be a qualifier for xlarge screen. Conversally, if you are designing for a small screen with MDPI you need to include small in the folder name
Screen Aspect	**long** **notlong**	This is to account for the aspect ratio of screens with the same dpi and different aspect ratios. **long:** Long screens; wqvga, wvga, fwvga **notlong:** not long screens; qvga, hvga, vga
Screen Pixel Density	**ldpi: 120 dpi** **mdpi 160 dpi** **hdpi: 240 dpi** **xldpi: 320 dpi** **nodpi: not dpi specific**	This accounts for the how the pngs are scaled, if they are scaled at all. If we use hdpi as the standard for designing, it breaks down like this. **ldpi: 2x less** **mdp: 1.5x less** **hdpi: 1** **xldpi: 1.5x more**
Dock Mode	**car** **desk**	This effects how the application reacts to car docking vs. desk docking
night Mode	**night** **notnight**	This effects how the application reacts to the time of day.

Qualifier	Value	
Touchscreen	**notouch** **stylus** **finger**	This is used to specify if the device has a touchscreen, stylus or not.
Language and Region	Examples: **en** **fr** **en-rUS**	If you are tailoring an app to a specific language, you need to add the language code. If it is regional it is comprimised of the language code and a region code. The region code is a 3 letter suffix starting with a r and ending in a country code. The list of country codes can be found at http://www.loc.gov/standards/iso639-2/php/code_list.php `and http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm
Screen Size	**small** **normal** **large** **xlarge**	This is your screen size. If you are designing an app to run on Honeycomb presently, there needs to be a qualifier for xlarge screen. Conversally, if you are designing for a small screen with MDPI you need to include small in the folder name
Screen Aspect	**long** **notlong**	This is to account for the aspect ratio of screens with the same dpi and different aspect ratios. **long:** Long screens; wqvga, wvga, fwvga **notlong:** not long screens; qvga, hvga, vga
Keyboard Availability	**nokeys** **qwerty** **12key**	nokeys: Device has no hardware keyboard qwerty: Device has a hardware qwerty keyboard 12key: device has a hardware 12-keyboard
Navigation Key Availability	**navexposed** **navhidden**	This decides whether the hardware navigation buttons are available to the user or not
Primary non touch navigation method	**Nonav** **dpad** **trackball** **wheel**	If your application is contigent on a certain type of navigation, these qualifiers are used to specify that
Platform Version (API Level	Examples **v3** **v4** **v7**	If your application is using functionality supported by a certain platform, you need to specify it.