Android Mobile Application Development from A to Z

an internet.com® Developer eBook

Contents…

[Android Mobile Application Development from A to Z]

This content was adapted from Internet.com's Developer.com website. Contributors: Shane Conder, Lauren Darcey and Keith Vance.

The Android Mobile Development Platform: A Reference Guide

By Shane Conder and Lauren Darcey

Android, an open source mobile platform with no upfront fees, has emerged as a new mobile development option that offers many benefits over competing platforms. But is it right for your project? In this reference guide, you'll learn all the nitty-gritty details you need to know to evaluate Android, including the tools and technologies for developing on the platform as well as the required costs. Armed with this information, you can make an informed decision as to whether or not Android is the right fit for your particular organization or development project.

Android Programming Languages

Native Android applications are written in Java. Applications requiring existing C/C++ libraries can take advantage of the Android Native Development Kit (NDK).

In addition to native Android applications written in Java, Adobe Flash and Adobe AIR support were added in Android 2.2, enabling a whole new group of developers to target Android devices.

Development Tools and Setup Costs

Unlike many mobile development platforms, Android is open and freely available. There are no developer fees or screening processes, nor must developers purchase expensive compilers or limit themselves to one specific operating system for development.

Android applications can be developed on a variety of operating systems, including:

- Windows XP (32-bit), Vista (32-bit or 64-bit), and Windows 7 (32-bit and 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Linux Ubuntu 8.04 LTS, Hardy Heron)

The Android SDK and development tools are freely available on the Android developer site, where developers can download the SDK after agreeing to the terms of the Android SDK License Agreement. Developers must also have JDK 5.0 or JDK 6 (freely available).

In terms of integrated development environments (IDEs), developers have a number of choices. Eclipse is the most popular IDE for Android development because it offers a handy Android Development Tools (ADT) plugin. At the time of writing, the ADT plugin supported both Eclipse 3.4 and 3.5. Developers can use other IDEs if they desire; the command-line tools that come with the Android SDK facilitate Android development and provide many of the features available within the Eclipse ADT plugin (e.g. creating projects, packaging resources and generating Android package files for deployment to devices, etc.).

Android Devices: Features, Functions and Availability

The only real cost for Android developers is the acquisition of device hardware. Although the Android team has insisted that testing within the Android emulator is generally sufficient for most development, we feel that emulators are no real substitute for testing on (at least some) target devices.

Fortunately (and unfortunately), many Android devices are available on the global market today. Consumers have an unprecedented number of choices in terms of distinctive devices, carriers, and payment plans. According to the official Google Blog, as of mid-2010, more than 60 Android handsets shipped from 21 different manufacturers. These Android devices are available on 59 carriers in 48 countries. In June 2010, Google announced that more than 160,000 Android devices are being activated each day (a rate of nearly 60 million devices annually).

Most Android devices are considered smartphones, with all the goodies one would expect (e.g. fast processors, touch screens, high-megapixel cameras, LBS services, accelerometers and so on). That said, other devices also run on the Android platform, including Internet tablets, e-book readers, TV boxes and others. It is certainly feasible to create a single application that can run smoothly across all these devices. However, developers will still need to identify and understand their target users and devices. Luckily, the Android platform and tools are designed to ensure maximum compatibility and to make compatibility a (relatively) straightforward matter for developers.

If you're unsure which Android devices to acquire for development purposes, consider one of the Android Dev phones, ADP1 or ADP2, which are available for purchase if you sign up as a developer to publish on Google's Android Market. The Android Dev phones are SIM-unlocked and therefore usable on any GSM network; they feature a bootloader that allows you to flash the device with different system images (helpful for mimicking various device platforms on a single device). Another SIM-unlocked handset is the Google Nexus One.

Android Development Framework and APIs

The Android application framework includes familiar programming constructs, such as threads and processes and specially designed data structures to encapsulate objects used by the Android operating system. With Android, developers use familiar class libraries exposed through Android's Java packages to perform common tasks such as graphics, database access, network access, secure communications and utilities. In addition to these familiar Java class libraries, such as java.net, developers can also rely on specialty libraries using well-defined open standards like OpenGL Embedded Systems (OpenGL ES), SQLite, and WebKit. The Android packages include support for:

- User interface controls (Buttons, Dropdowns, Text Input, Grids, Tabs, Gallery)
- Flexible user interface design and layout (programmatically or via XML)
- Secure networking and Web-browsing features (SSL, WebKit, XML parsing)
- Structured storage and relational databases (SQLite)
- Powerful 2D and 3D graphics (including OpenGL ES 2.0
- Enhanced support for audio, still images, and video media in many formats, "ducking," etc.
- Access to underlying hardware sensor data, the camera, accelerometer, etc.
- Access to underlying services like location-based services (LBS), Wi-Fi, Bluetooth, etc.
- A robust unit testing framework for automated testing of Android apps

One of the Android platform's most compelling features is well-designed application integration. Developers can write applications that integrate seamlessly with other Android applications, including core platform applications such as the Browser, Maps, Contacts, Messaging and Email.

On the Android platform, all apps are created equal. There is no distinction between native and third-party applications, enabling healthy competition among application developers. All Android applications use the same libraries and have access to the same permissions options and functionality. Android applications have direct access to the underlying hardware, allowing developers to write much more powerful applications.

The Android SDK also comes with extensive developer documentation. Developers can also find the complete documentation online with videos, the official Android developer blog and an active Android development community.

Publication Opportunities and Target Audience

In terms of market share, the Android platform has been gaining ground steadily against competitive platforms such as the Apple iPhone, RIM BlackBerry and Windows Mobile (all of which have been around considerably longer). The latest numbers from The Nielsen Company (as of Summer 2010) show BlackBerry in the lead with 35% of the smartphone market, and declining. Trailing close behind and gaining is Apple's iPhone at 28% and Microsoft Windows Mobile is declining with a 19% showing. Android is trailing with 9%, but its growth numbers are accelerating rapidly and according to some sources Android devices are selling faster than most, if not all, competing platforms. If you look back over the past 18 months since Android first became available to consumers, you can see that the platform has been gaining ground steadily at the expense of its competitors, yet the market could potentially accommodate substantially more growth by the platform.

Android applications have none of the costly and time-intensive testing and certification programs required by other platforms. Android developers are free to choose any kind of revenue model they want. They can develop freeware, shareware or trial-ware, ad-driven applications, and paid applications. With Android, developers can write and publish any kind of application. Developers can tailor applications to small demographics, instead of just the large-scale money-making ones often insisted upon by mobile operators. Vertical market applications can be deployed to specific, targeted users.

Because Android developers have a variety of application-distribution mechanisms to choose from, they can pick the methods that work for them instead of being forced to play by others' rules. Android developers can distribute their applications to users in a variety of ways. The most popular distribution mechanism is Google's Android Market. The Android Market is a generic application store with a revenue-sharing model. As of July 2010, more than 90,000 applications were available in the Android Market. Many other Android applications have been sold through other publication channels. More than 180,000 Android developers have downloaded the Android SDK and developed Android applications.

Pros	Cons
Apps written in Java, which is an easy, commonly understood programming language	If you don't like Java, you're out of luck.
Very low barrier to entry. No vetting of developers to determine whether they are worthy of developing apps. No fees to join development community.	Developer expertise and application quality will vary greatly. Neither apps nor developers are "curated" as they sometimes are on other platforms, but technically, malicious developers exist on all platforms.
Freely available development tools are popular, powerful and generally well designed. In fact, the Android development tools are impressive compared to others provided for other mobile platforms.	Like all mobile platforms, developers are reliant on the platform and tool developers to address tool defects and limitations.
	Continued to next page...

Android SDK and operating system is powerful, full of features, and easy to use. There is no distinction between developer apps and native apps. Developers have unprecedented access to underlying hardware on device in a safe and secure manner.	With this power comes greater developer responsibility to design stable, responsive applications.
Many publication channels available, including publishing through consolidated app markets as well as self publication. Suitable for mass market, enterprise development and everything else.	When developers choose to publish through third-party app markets, they often must give up a portion of their profits. Because there are many publication channels, software piracy becomes a bigger concern.
Vast variety of devices available throughout the world, on many carriers, with many plans, including many with unlimited data.	Device variety (aka fragmentation) makes compatibility something that developers must address (various screen sizes, optional hardware features, etc.).
Rapidly growing market share. There's still room for lots of growth. Applications can still distinguish themselves, and killer apps are developed all the time.	Still smaller market share than more mature mobile platforms such as BlackBerry and iPhone.
Free, open platform allows any hardware manufacturer to build devices based on the open platform.	The freedom to grab the open source Android platform and place it on any device -- without anyone's permission -- can lead to incompatible devices, confusion among users, and proliferation of platform devices and distribution mechanisms for developers to support.

Summary

Android developers enjoy many benefits over competing mobile platforms. Android is relatively new to the mobile scene, offering a distinctly different approach: it's open and free with robust access to the underlying device hardware. Designed in Java, Android applications can be created on a variety of operating systems with free and readily available tools. Getting started with writing mobile applications has never been so easy or affordable. And after you've created those apps, you can rest assured that there are millions of Android users all over the world who are ready to use them. ■

Top 10 Features for Developers in Android 2.2

By Shane Conder and Lauren Darcey

Android 2.2 (codename: Froyo) was a minor SDK release, but it still packed some punch, providing both developers and users with some much-anticipated features. After attending the Google I/O 2010 conference in May and witnessing the Froyo announcement, here are the top 10 features (in no particular order) that we think developers will benefit from most.

1. Flash 10.1 and AIR Support

There may be some disagreement about the viability of Flash on mobile, but it's coming to Android phones. Whether or not Flash is the future is really not the question; for now, Flash is pretty pervasive on the Web, so cutting it out is in effect cutting out many of the dynamic Web apps users already enjoy.

Beginning with the Froyo release, Android users will be able to download the open beta version of Flash 10.1 as well as AIR support (in the form of an Android application) from the Android Market. This decision substantially expands the number of Web applications and sites accessible to Android users and widens the development community for Android substantially.

This may become a double-edged sword for Android developers, however. How will this change the content of the Android Market? With boatloads of Flash apps out there, why would anyone want to bother creating a native Android app version? Well, there certainly are valid reasons, but we think a lot of companies would need to be convinced when native apps require them to target multiple platforms to reach their customers. Maybe Flash apps will help weed out the weak and badly written native Android app competition, but will it strengthen the Android development community as a whole? We'll have to wait and see.

2. Push Messaging

Developers can now leverage another of Google's services, the Android Cloud to Device Messaging (C2DM) framework. This framework provides a service for enabling limited push functionality to Android devices through Google services, which handle the queuing and secure delivery of lightweight messages to the device. While the framework is getting ironed out, developers can sign up at the Google Labs website. Google apps, such as the Android Market for the Web, will soon be using this feature to push Android applications purchased via the Web to the phone over the air. This technology should help resolve some of the crazy polling traffic caused by Android applications at the moment (resulting in reduced battery life, performance reductions, etc.).

3. New Enterprise Features

Android is finally positioning itself for some serious enterprise use. The Android 2.2 SDK includes new Device Administration APIs (android.app.admin) for remote and secure device management. Here you'll find APIs for managing device security, including password policy enforcement and the ability to remotely lock and wipe the device. For example, if an employee lost his or her device with its sensitive data and credentials, it could quickly be locked and wiped of that data.

Froyo also introduces more robust Microsoft Exchange support (see No. 9). We also heard some murmurs about Android Market-like deployment solutions for the enterprise, but these will likely come later (but not in Froyo).

4. Performance Improvements

Developers and users benefit from Froyo's vast and deep performance improvements. It seems some of Google's most bloodthirsty quality and performance geeks combed the platform for "jankiness" (a term heard often from Googlers that means uneven performance and responsiveness) and built in a lot more instrumentation and benchmarking behind the scenes. This much-needed performance overhaul resulted in a smoother, leaner platform that hums — and a plan to keep it lean and speedy in the future.

This is partly due to the inclusion of a just-in-time (JIT) compiler for the Dalvik VM. According to the Google Android team, Froyo runtime performance is 2 to 5x faster than previous versions of the Android platform. You can disable JIT optimization within the application's Android Manifest file. The Android browser is also noticeably faster due to its V8 JavaScript engine, resulting in a 2 to 3x boost in performance compared to Android 2.1's browser (also see No. 10).

5. Audio and Media API Improvements

A number of problems and logistical issues with the Android media APIs have been addressed with the Froyo release. For example, an Audio Focus API has been added for managing audio playing etiquette amongst competing applications. The SoundPool API was also updated to include a callback for when an item has completed loading, as well as the ability to pause and resume all streams so an application no longer needs to keep track of each nor do these actions on each individual stream. These improvements ease implementation and enhance efficiency.

6. Across-the-Board SDK Enhancements

Numerous Android APIs were added as part of the 2.2 release. Graphics and game developers will welcome the support for OpenGL ES 2.0 and ETC1 texture-compression support. Services like speech recognition (android.speech) received substantial upgrades and peripheral APIs such as those that support the Camera and Camcorder have been greatly improved. A new UI Mode Manager (android.app.UIModeManager) service adjusts the device configuration for night mode, car mode, and desk mode (docking state).

As of Android 2.2, applications are not limited to installation paths on the main device, but can also be installed on external storage such as an SD card. There is also a new generic data backup service Android applications can use to allow users to transition seamlessly between Android devices.

In terms of sensible but frustrating API changes, the layout attribute fill_parent has been renamed to match_parent (no, it won't break your old apps — yet). Even debugging got an enhancement with a blob-based "logcat" style queue of data in the form of DropBoxManager. The list goes on.

7. Android Market Updates

There are quite a few updates to the Android Market coming with the Froyo release. One of the most useful new features for publishers is built-in error reporting. If your application crashes on a user's phone, the user will have the option to send the error report back to the specific publisher through the Android Market. This enables a full-circle, user-developer feedback loop, allowing publishers to address problems (and receive valuable crash diagnostic information, such as the device configuration and stack trace) and avoid ratings disasters.

And let's not forget that Android developers are also Android users. In addition to the expected performance improvements and "Chrome" added for the Froyo platform release, there are a number of compelling consumer features delivered in Froyo, such as:

8. Tethering and the Portable Hotspot

Froyo delivers USB tethering and the ability to turn your Android device into a portable Wi-Fi hotspot. This is a an awesome feature, but we wonder how many operators/carriers are going to hide and/or disable it, as some have done with similar features on competing platforms. If we had to guess, we'd say that this is one of those compelling features that might improve Android adoption (although at 100,000 device activations a day, Android isn't doing too badly) but also is most likely to frustrate these same people when they cannot get that feature on a shipping phone. We'll see.

9. Microsoft Exchange Support

Nobody likes juggling devices based on whether they're using it for work or personal reasons, and there's a pretty substantial class of users out there who are greatly limited in their choice of mobile devices based upon their corporate IT requirements — notably, support for Microsoft Exchange. (Frankly, it seems crazy to us to call anything a "smartphone" if it doesn't support Microsoft Exchange, but that's just us.)

Android 2.2 includes lots of new Microsoft Exchange features, such as:

- Improved security features allowing administrators to enforce password policies
- For Exchange administrators, remote wiping of a device if it is lost or stolen
- Exchange calendar support now compatible with the Android Calendar app
- Auto-discovery for easy account setup and syncing
- Android email support for auto-completion of recipient names and addresses using Microsoft Exchange Global Address Lists

10. "The World's Fastest Mobile Browser"

In the Google I/O keynote, Google VP of Engineering Vic Gundotra made the claim that the Android 2.2 release includes the world's fastest mobile browser. He illustrated this claim by basically "lapping" the iPad browser (even after giving it a head start) in a little race based upon SunSpider JavaScript benchmarks. Whether or not the demonstration was a fair apples-to-apples comparison, the point got across: the performance improvements made to the Froyo browser made it wicked fast. Android browser performance makes everyone happy (OK, maybe not competitors …) and will surely be appreciated as HTML5 grows up. ■

Building Your First PHP for Android Application

By Keith Vance

Google's open source Android mobile operating system is taking the smartphone market by storm. Unlike Apple, which has stringent guidelines and requirements for developers who want to offer their applications on the iPhone App Store, Google has left the Android platform wide open. You can even write Android applications in PHP now. The folks at Irontech have created a PHP port to run on Android, and with the Scripting Layer for Android (SL4A), you can build PHP Android applications.

In this article, I'll explain how to install, set up and use PHP for Android and SL4A, I'll present a demo application as an example, and I'll give a first-hand account of the PHP for Android developer experience.

Installing PHP for Android

To install PHP for Android, you have to have a phone or emulator running Android version 1.5 or higher and you must enable "Unknown Sources" under Application settings. After you have that set, you simply install the SL4A environment and the PHP for Android APK.

Installing the SL4A is straightforward, but after you install the PHP for Android application, you need then click "install" again for it to be fully installed and functioning. If you have trouble with the installation, there's a handy video demonstration available on Vimeo to walk you through the process.

Setting Up the PHP for Android Development Environment

If you installed PHP for Android, theoretically, you can write PHP Android applications with your phone. But for all practical purposes, that doesn't work very well. What you should do is download the Android SDK, set up an emulator and write code using your favorite editor.

After you've downloaded the SDK, extract it in a directory of your choosing, run the Android application located in the tools directory, and set up an emulator. From the *Android SDK* and AVD Manager menu, select Virtual Devices and click the New button. Name your new emulator (e.g. "Droid2") and select Android 2.2 as the target. Enter 10 MiB for SD Card size and click *Create AVD*.

Now that you've got the Droid emulator set up, click the Start button. Here's where things get a little tricky, because you can't just copy files to the virtual device you just set up. You have to set up port forwarding and push your PHP script to the virtual device using a program called *adb*, which is part of the Android SDK. It is located in the tools directory too.

Next, you will start a server on the virtual device. You will connect with this server to send your scripts. The following steps will get you up and running as quickly as possible (you can read the full documentation for this process at code.google.com/p/android-scripting/wiki/RemoteControl).

1. With your new virtual device running, go to the *Applications* screen and click SL4A.
2. In the SL4A screen, click the *Menu* button, select View and choose *Interpreters.*
3. Click *Menu* again, select Start Server and choose *Private.*
4. Drag the Android notification bar down and you should see SL4A Service. (Click the service and note the port number your server is listening on, e.g. 47000.)
5. Open up a shell or command prompt and set up port forwarding using the *adb* tool. For example, enter the command adb forward tcp:9999 tcp:47000, replacing 47000 with your port number.
6. Set up the AP_PORT environment variable. On UNIX or Mac, run export AP_PORT=9999. On Windows, type set AP_PORT=9999.
7. To test your script with your emulator, just run adb push my_script.php /sdcard/sl4a/scripts, replacing my_script.php with the script you wrote.

You can also set this up to work with an actual phone. Just follow all of the steps you did with your emulator on your phone. To make things easier, you also should set up an ANDROID_HOME environmental variable that points to your Android SDK location and add the tools subdirectory to your path.

Building an Android Application with PHP

Writing a PHP application to run on Android is really pretty simple after you set up your development environment. One thing you'll notice is that the version of PHP included with PHP for Android is an extremely stripped down build. You basically have the core PHP functions and JSON support — that's about it. And if you're an Android developer who's familiar with the Java framework, you'll notice that the Scripting Layer for Android doesn't provide access to all of the components you're used to having when building a full-blown Android application with Java.

What SL4A does provide are "facades" to a subset of the Android APIs. (A complete listing of all of the methods available via the SL4A is available at code.google.com/p/android-scripting/wiki/ApiReference.) But what's fun about PHP for Android is that you can quickly prototype an application and see it in action with just a few lines of code. I'll demonstrate this with an application for checking stock quotes that's less than 60 lines of code.

Copy and paste the code in Code listing 1 at the end of this article into your editor and save it as quoter4android.php and upload it to your emulator. If your emulator isn't running, fire it up, configure your port forwarding and upload the quoter4android.php file with the adb application included in the Android SDK tools directory.

To run the application in your emulator, go to the Applications screen, click the *SL4A* icon and click the *quoter4android.php* option.

To install *quoter4android.php* on your phone, you can set up port forwarding. But it's easier to just plug the phone into your computer via USB and copy the script into the sl4a/scripts directory. However, to run scripts on your phone, you have to unplug it from your computer or else you won't see any of the installed scripts when you click on the SL4A icon.

You'll notice that the first line of this application sets up a constant QUOTE_SERVER. If you're used to building traditional PHP Web applications, you don't have to worry about distributing your code and making changes to it in the future -- that's not how it works with Android. You have to distribute your actual PHP code. So if you decide to put your PHP Android application in the Android Market and you hardcode a Web address in it that you don't control, your application could break down the road.

For example, this stock quote application actually pulls the stock information from a Yahoo Web service. But rather than hardcoding direct access to Yahoo into the Android application, I created a simple Web service as a link between the Android application and the Yahoo stock service. So now if Yahoo decides to stop offering this service, or if they change the way it's accessed, I can just update my Web service located at *quoter.take88.com*. The Android code doesn't need to change, and nobody's walking around with a broken application on their phone. Also, by leveraging a Web service, I was able to take some of the complexity out of the Android application and move it to my Web service, where I have access to full-blown languages and not just a stripped down version of PHP. In this case, I wrote the Web service in Perl using mod_perl.

Conclusion

There's a lot you can do with the SL4A and PHP for Android; this article only scratches the surface of what's possible. While both projects are very young — in fact, a new version of SL4A dropped while I was writing this story (feel free to run the newest version) — as they mature, more possibilities will present themselves. In any case, keep your Android applications small, tight and light. ■

Code Listing 1. quoter4android.php

```
<?php
define('QUOTE_SERVER', 'http://quoter.take88.com/?ticker=%s');
require_once("Android.php");
$droid = new Android();
$action = 'get_tickers';
$tickers = '';
while (TRUE) {
    switch ($action) {
    case 'quote':
       $droid->dialogCreateSpinnerProgress("Querying stock information server ...", "Please wait");
       $droid->dialogShow();
       $quotes = @array_slice(json_decode(file_get_contents(sprintf(QUOTE_SERVER, $tickers))), 0, 3);
       $droid->vibrate();
       $droid->dialogDismiss();
       // Possible data points.
       // "SYMBOL","NAME","LAST_TRADE","MORE_INFO","LAST_TRADE_DATE","LAST_TRADE_TIME","OPEN","DAYS_
HIGH","DAYS_LOW","DIVIDEND_SHARE","PE_RATIO","52_WEEK_LOW","52_WEEK_HIGH","VOLUME"
       $output = '';
       for ($i = 0, $cnt = count($quotes); $i < $cnt; $i++) {
           $output .= "Company: " . $quotes[$i]->NAME ."\n";
           $output .= "Ticker: " . $quotes[$i]->SYMBOL . "\n";
           $output .= "Last trade: $" . $quotes[$i]->LAST_TRADE . "\n";
           $output .= "\n";
       }
         $output = html_entity_decode($output, ENT_QUOTES, "UTF-8");
       // Something is wrong with &apos;
       $output = str_replace("&apos;", "'", $output);
       $droid->dialogCreateAlert("Your stock quotes", $output);
```

```
        $droid->dialogSetPositiveButtonText("Get new quote");
        $droid->dialogSetNegativeButtonText("Exit");
        $droid->dialogShow();
        $response = $droid->dialogGetResponse();
        if ($response['result']->which == 'negative') {
            $action = "exit";
        } else {
            $action = 'get_tickers';
        }
        break;
    case 'get_tickers':
        $response = $droid->getInput("Stock Tickers (max. 3)", "Enter Tickers.\nSeparate with 
spaces.");

        $tickers = str_replace(' ', '+', $response['result']);
        //      print_r($response);
        //$tickers = $response['result'];
        //      print_r($tickers);
        $droid->vibrate();
        $action = 'quote';
        break;
    case 'exit':
        $droid->exit();
        exit();
        break;
    }
}
?>
```

Building Killer Android Tablet Apps: Design and Development Tips

By Shane Conder and Lauren Darcey

Device manufacturers are ramping up an exciting new line of Android devices: tablets. The success of the Apple iPad has proven that consumers are ready for these devices, which make consuming media content like video and audio a rich and enjoyable experience. But there's a catch: until now, Android developers have made certain assumptions about the target devices their apps run on — assumptions like "the device is a phone," "the device has a small screen" and "the device includes the Google app experience." These assumptions will not always hold true for tablets and other types of Android-powered devices. In this article, we offer some tips and tricks for ramping up your skills to design and develop killer apps for the Android devices of the future.

Android Supports Tablets?

Yes, and no. You may be aware that Android has been ported to many kinds of devices, including phones, toasters, microwaves and laptops. However, just because Android runs on these devices doesn't mean the user experience is great or the device is officially recognized by the Android community.

So what about the Android tablets that are already in users' hands? Well, the Android operating system is open and free. Manufacturers can put Android on whatever devices they want to, and many have. Archos has been making Android tablets for quite some time. However, these devices run a modified version of the Android OS that has been tuned for the tablet device. Until now, tablets have been something of a gray market, but that's about to change.

Until recently, Google and the Open Handset Alliance have not approved any Android tablets for use with Google proprietary applications such as Gmail, Maps and most importantly the Android Market. This will change with the next wave of Android tablets; Google has acknowledged that tablets and other devices will be recognized and incorporated into the Android platform in future versions of the Android SDK and the Android Market. Now, there's a wave of new Android-powered devices slated to hit the shelves late this year and early next year from the likes of Acer, Dell, Samsung, Toshiba, Viewsonic, Archos and more.

Developers are eager to write apps for these exciting new devices and ensure that their existing apps will run smoothly. The question is: how? Google has made a statement to the effect that the current version of Android (2.2, or Froyo) is not designed for tablets. The next version of Android (Gingerbread) is likely to address some of these issues, but developers need not wait to start preparing for the onslaught of Android tablets.

Application Design for Android Tablets

Lazy development assumptions may have worked when there was really only one type of device (a phone), but these bad habits may come back and bite you when your app is deployed on a device like a tablet. Reconsider previous design decisions now and update your applications to make them compliant with the latest configuration options available on the Android platform to help ensure that your application is ready for the future.

The good news is that developing for new Android devices isn't going to be that different from developing for existing ones. Most existing apps will run well enough, provided they've been designed prudently, by which we mean:

- The app properly identifies its application hardware and software requirements using the Android Manifest file tags such as supports-screen, uses-configuration, uses-feature and uses-permission.
- The app code checks for hardware, services and optional APIs before attempting to use them.
- The app designers minimized the assumptions about which exact devices or hardware the application would run on.

Just as not all Android devices support Bluetooth or WiFi, there are — and will continue to be — new optional APIs for working with specific devices, including tablets. Some of these APIs may be baked into future versions of the Android SDK (like Gingerbread) while others may be third-party add-ons available from manufacturers. These may be similar to such add-ons available for current handsets; the SenseUI is available for some but not all HTC devices, or MotoBlur on some but not all Motorola devices.

User Interface Design for Tablets

When it comes to designing user interfaces for tablets, it's best to stick with flexible layout designs that will scale well to various screen sizes, resolutions and orientations. This way, users will find the experience familiar, regardless of what type of device they use. Here are some tips for designing user interfaces for tablet devices:

- Keep screens streamlined and uncluttered and ensure touch controls such as buttons are of adequate size.
- Use flexible layout controls such as LinearLayout and RelativeLayout as opposed to pixel-perfect ones such as AbsoluteLayout.
- Use flexible dimension values like dp and sp instead of px or pt.
- Use alternative resources such as graphic and dimension resources to provide specialized resources for different screen sizes, aspect ratios, pixel densities and touchscreen types.
- Use alternative resources such as layout and graphic resources to provide specialized resources for landscape and portrait modes.

Testing Apps for Android Tablet Compatibility

Although few tablets have been released yet, nothing is stopping you from beginning to test your existing apps and ironing out the obvious issues. Here are some tips to keep in mind when testing for tablet compatibility:

- Testing on the actual devices (as opposed to the emulator) will be critical to ensuring your application behaves as expected. Some devices, such as Samsung's Galaxy Tab, have reported their hardware characteristics differently than expected. For example, despite having a medium-density screen, Samsung chose to have its new tablet report as a high-density screen because it looks better (see Figure 1).
- Some tablets may not include the Google "experience," so make sure you also test with Android Virtual Devices (AVDs) that do not include the Google add-ons.
- Tablets, among other devices, are beginning to take a landscape-first approach to the screen. Make sure your app displays properly in both orientations and handles orientation changes correctly.

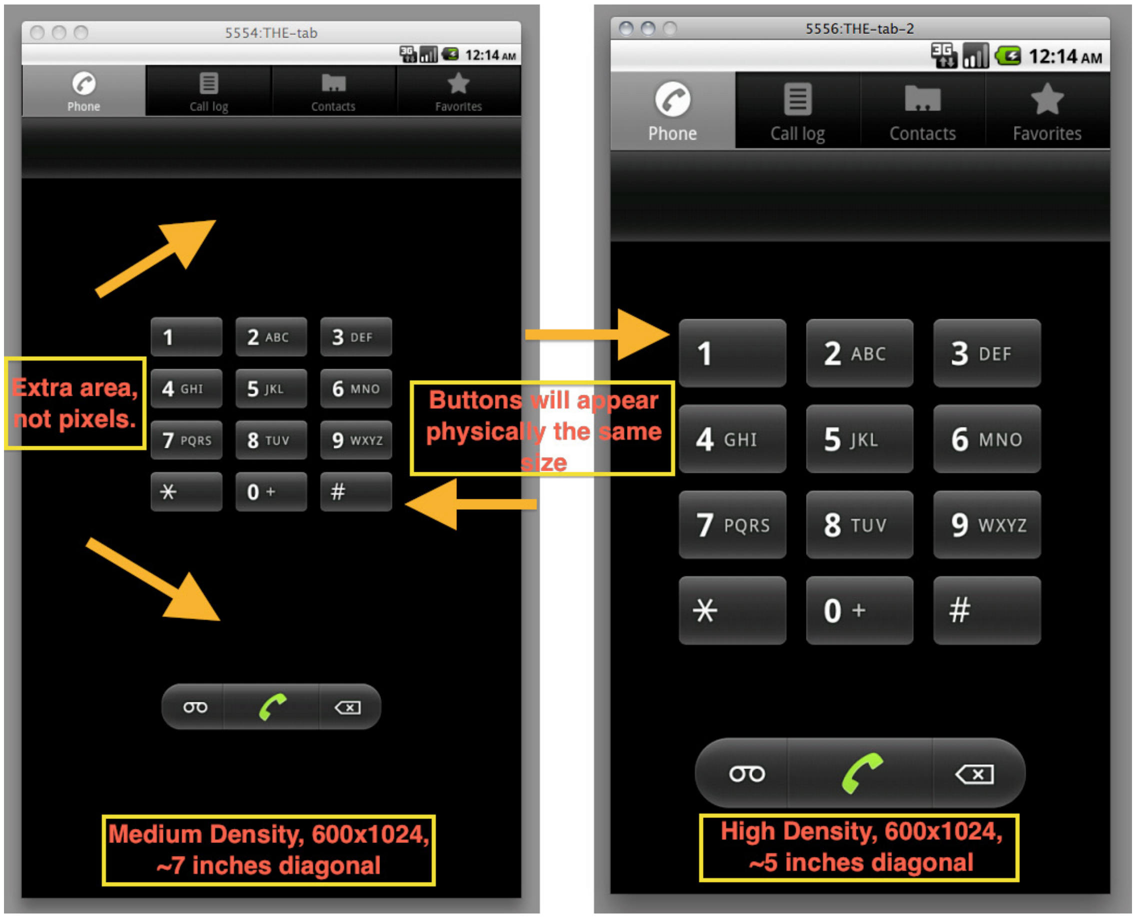
5554:THE-tab
12:14 AM
Phone
Call log
Contacts
Favorites
Extra area, not pixels.
Buttons will appear physically the same size
Medium Density, 600x1024, ~7 inches diagonal
5556:THE-tab-2
High Density, 600x1024, ~5 inches diagonal

Figure 1. Android App Mimicking a Tablet on a Custom AVD

Finally, one of the best things you can do right now to ensure your app is tablet compliant is load your app into the emulator with a tablet-style AVD configuration and see how it behaves. For example, use the following steps to create an AVD configuration that mimics how your application would display on a tablet much like the upcoming Galaxy Tab:

1. Launch the Android SDK and AVD Manager.
2. Press the *New…* button.
3. Enter a name for the tablet (e.g. "Tablet Emu") and choose an appropriate SDK version, such as Android 2.2.
4. Create an SD card (we use between 32MB and 512MB).
5. For the *Skin* section, choose Resolution and enter "1024" and "600" into the appropriate boxes. If you enter 1024 then 600, the device will start in landscape mode. If you enter 600 then 1024, it will start in portrait mode.
6. For the Abstracted LCD Density, any value will work (although the values 120, 160 and 240 are suggested). To mimic the actual screen density of the Galaxy Tab, enter 170. The device will be treated as a medium-density display. To mimic the reported screen density, enter 240. The device will be treated as having a high-density display.
7. Choose *Create AVD.*

8. When it's created, launch it with the *Start…* button.
9. As the display size is rather large, you may wish to scale it down using the launch parameters.

Figure 1 illustrates how an application might appear on a custom AVD to mimic a tablet. It also demonstrates the difference between a medium-density display with the same pixel resolution as a high-density display. Because the density is different, the buttons on each screen actually draw at the same size. The medium-density display, however, shows a lot of wasted screen space. This not only demonstrates one example of why Android isn't quite ready for larger screen tablets, but also shows why a manufacturer might want to report a different value.

If the rumors are true, the next major release of the Android SDK (Gingerbread) will begin to address some of the device differences in some official manner. Expect to see changes such as additional APIs for optional hardware, updates to the Android Manifest configuration options available for targeting specific device characteristics and perhaps new controls and screen layout options. We are also likely to see changes to the Android Market to reflect the plethora of devices about to reach consumers' hands. For example, sources at Google have implied that certain application permissions (as defined in the Android Manifest file using the tag) may be used by the Android Market to filter apps for devices in the future.

In that future, Android-powered devices will likely come in many forms: phones, PDAs, music players, tablets and toasters. For now, one of the best things you can do as a developer is start to think along these lines. Be mindful of the assumptions you make when developing your apps, and consider how they will restrict or allow your apps to run on different types of devices. Review your existing apps and update them with more flexible user interfaces and prudent assertions on device features and characteristics.

With the introduction of Android tablets, developers now have a whole new range of devices to target with their applications. Android tablets are likely to boast larger and higher-resolution touchscreens, video output options, front-facing cameras and other optional hardware features — at very reasonable prices. These features enable developers to write new kinds of applications and enter new markets. Developing Android apps for tablets requires some forethought, but many of the design principles for writing great Android apps for tablets really apply to all device targets. ■