

# AT&T Developer Program

## Developing Applications for Android

### White Paper

Document Number	<b>1.0</b>
Revision	<b>0.6</b>
Revision Date	<b>04/08/2010</b>

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You solely on the basis that You will be responsible for making Your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

## Revision History

© 2009 AT&T Intellectual Property  
All rights reserved.

AT&T and AT&T logos are trademarks of AT&T Intellectual Property.

All marks, trademarks, and product names used in this document are the property of their respective owners.

Date	Revision	Description
02/12/2010	0.1	This document is currently at the Draft stage.
02/21/2010	0.2	This document is currently at the Draft stage.
02/22/2010	0.3	This document is currently at the Draft stage.
03/07/2010	0.4	This document is currently at the Draft stage.
03/14/2010	0.5	This document is currently at the Draft stage.
04/08/2010	0.6	Final

**AT&T Developer Program ..... 1**

**White Paper ..... 1**

**All marks, trademarks, and product names used in this document are the property of their respective owners. .... iii**

**Figures ..... vi**

**Tables ..... vi**

**1. Overview ..... 7**

    1.1 Android Overview ..... 7

    1.2 Android Architecture ..... 8

    1.3 Available Libraries (Available APIs)..... 10

    1.4 Comparing Android to Other Platforms ..... 11

    1.5 Terms and Acronyms..... 13

    1.6 Why Develop for Android..... 14

**2. Developing Applications..... 15**

    2.1 How Android Executes Applications..... 15

    2.2 Application Components..... 16

    2.3 The Android Manifest File (AndroidManifest.xml) ..... 22

    2.4 Activating Application Components Using Intents ..... 25

**3. Developing App Widgets ..... 29**

    3.1 Creating App Widgets..... 29

    3.2 Layouts and Views..... 30

    3.3 Handling Events..... 30

**4. Communications..... 33**

    4.1 Basic Networking ..... 33

    4.2 Connections and Management..... 35

    4.3 Telephony ..... 39

**5. Android UI ..... 48**

    5.1 Layouts ..... 48

    5.2 UI Components..... 52

    5.3 Menus ..... 55

    5.4 Events ..... 56

- 6. Writing for the Web..... 59**
  - 6.1 Mobile Web/XHTML Sites..... 59
  - 6.2 Leveraging HTML5 ..... 60
  - 6.3 Extending the Browser..... 61
  
- 7. Best Practices ..... 64**
  - 7.1 User Interface ..... 64
  - 7.2 Optimizing Code for Android ..... 66
  - 7.3 Localization ..... 67
  - 7.4 Common Issues..... 69
  
- 8. Android Development Tools and Android SDK..... 71**
  - 8.1 Creating an Android Project ..... 71
  
- 9. Deployment ..... 75**
  - 9.1 Signing Your Application..... 75
  
- 10. Porting Applications to Android ..... 77**
  - 10.1 J2ME Applications ..... 77
  - 10.2 Going Native ..... 78
  
- 11. Conclusion ..... 79**
  
- 12. Works Cited..... 80**

## Figures

Figure 1 — Android Platform Architecture. Source: Google. ....	9
Figure 2 — Some examples of App Widgets running on the Android home screen. Courtesy Google. ....	29
Figure 3 —The Android view hierarchy.....	48
Figure 4 — A LinearLayout set up in the vertical orientation. All three children above are ‘stacked’ one atop the other in a single column. ....	50
Figure 5 — A Table layout .....	50
Figure 6 — A Relative layout .....	51

## Tables

Table 1 — An overview of the Android libraries. Source: Google. ....	10
Table 2 — Android SDK compared to other popular mobile SDKs. Source: Adapted from Engadget. ....	12
Table 3 — Terms and Acronyms .....	14
Table 4 — Default action strings supported by Android. ....	27
Table 5 — Default intent categories supported by Android. ....	28
Table 6 — Networking interfaces in Android.....	33
Table 7 — Android networking APIs .....	34
Table 8 — Parameters for setting up a network connection .....	35
Table 9 — Public methods for obtaining network information .....	38
Table 10 — Public methods for network event notification .....	38
Table 11 — Android telephony methods.....	40
Table 12 — Parameters for generating SMS messages .....	46
Table 13 — A partial list of the UI widgets available in Android. ....	55
Table 14 — Event listener interface callback methods .....	57
Table 15 — HTML5 features supported in Android .....	60
Table 16 — WebView customization points.....	62
Table 17 — Sample performance times of some common actions .....	67
Table 18 — Default resources to define for localization .....	68
Table 19 — Project files automatically generated by the ADT New Project Wizard.....	73
Table 20 — SDK tools needed when using IDEs other than Eclipse + ADT .....	74
Table 21 — Items generated for a new project using the Android tool.....	74

## 1. Overview

Times have never been better for mobile software developers. With the introduction of Android from Google and the Open Handset Alliance, developers have a robust platform to run their applications, and extend their reach to millions of mobile users.

This paper will introduce you to the Android platform and its capabilities. It will give you an overview of developing applications for Android, as well as tips and tricks for optimizing your applications and launching them on the AT&T network.

This paper is aimed at experienced software developers who are new to the Android platform.

The main topics of this paper are:

- Developing Applications,
- Developing App Widgets,
- Communications,
- Android UI,
- Writing for the Web,
- Best Practices,
- Android Development tools and Android SDK,
- Deployment, and
- Porting Application to Android.

### 1.1 Android Overview

Android is an operating system for mobile devices developed by Google and the Open Handset Alliance. The Android operating system supports smartphones

similar to Apple iPhone, Palm webOS, RIM BlackBerry, Windows Mobile, BREW MP, and others.

Android was initially developed by Android, Inc., a small California startup which was later purchased by Google. The initial goal of Android, Inc. was to create an open and extensible operating system for mobile devices, which would give mobile carriers flexibility to create unique devices for their networks. When Google bought Android, Inc. in 2005, the industry assumed Google would launch its own mobile device. However, Google continued development on Android and helped form the Open Handset Alliance in 2007, which is a consortium of 47 hardware, software and telecom companies.

Open Handset Alliance (OHA) launched the first Android handset in 2008. By the end of 2009, more than 20 Android handsets were launched, and in January 2010, Google launched its own handset, the Google Nexus One.

## 1.2 Android Architecture

The Android operating system is comprised of a virtual machine that runs on the Linux kernel, plus APIs, and a collection of built-in applications. The built-in web browser is built on the WebKit layout engine, also used by iPhone, Palm webOS and some Symbian devices, in addition to Google's desktop browser, Chrome.

Android runs applications within virtual machines (VMs). The highly-optimized virtual machine implementation (called the Dalvik virtual machine) adds a level of security to the operating system, since applications can be kept separate from the main part of the operating system, and from each other. To run multiple applications simultaneously, Android runs multiple virtual machines, and can do so even on devices with limited memory (RAM).

### 1.2.1 Hardware Designs

Android devices are made by a number of manufacturers, and designs vary widely. The Android platform supports a variety of hardware features including accelerometers, displays of varying sizes including multi-touch capable screens, physical keyboards, cameras, and a variety of cellular radios. Android is being used for other devices as well, including the Nook e-book reader from Barnes & Noble, as well as some upcoming tablet computers.

### 1.2.2 Software Platform Architecture

The diagram below gives a high-level view of the architecture of the Android software platform.

Android is comprised of:

- **Built-in Applications** for messaging, web browsing, calendaring, etc.
- **An Application Framework** to write new applications,
- **Libraries** to access device functions from within applications,
- **The Android Runtime** to run built-in and user-installed applications,
- **The Linux Kernel** for core services such as graphics, security, and networking, and memory management.



Figure 1 — Android Platform Architecture. Source: Google.

### 1.3 Available Libraries (Available APIs)

Android includes a set of C/C++ libraries used by various components of the Android system. Developers can access the functionality of these libraries via the Android Java APIs exposed in the Android SDK. The table below lists some of the core libraries in the Android 2.x SDK (updated January 2010).

Library	Description
System C Library	For general application development. An implementation of the standard C system library (libc), based on BSD and tuned for Linux embedded devices
Media Libraries	Provides media playback and recording services, and still image display services. Supports many popular formats such as MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG. Based on PacketVideo's OpenCORE
Surface Manager	Manages access to the display subsystem and composites 2D and 3D graphic layers from multiple applications
LibWebCore	Provides web rendering services. LibWebCore is based on WebKit, and powers the Android browser and an embeddable web view
SGL	Provides 2D graphics rendering services
3D Libraries	Provides hardware-accelerated 3D graphics rendering services, as well as a highly optimized software rasterizer used when hardware acceleration is not available. Based on OpenGL ES 1.0
FreeType	Provides font rendering for both bitmap and vector-based fonts
SQLite	Provides relational database services, available to all applications

Table 1 — An overview of the Android libraries. Source: Google.

## 1.4 Comparing Android to Other Platforms

Table 2 is a high-level comparison of the Android platform to competing mobile/embedded platforms.

Category	Android	BlackBerry	BREW MP	iPhone	J2ME	Symbian S60	Windows Phone 7
<b>Development Language(s)</b>	Java, C++	Java	C/C++, Java(J2ME)	Objective-C	Java	Java, MIDP, C++, Python, Adobe Flash	C++, C#, VB.NET, C++
<b>Native Development</b>	Yes, C/C++	No	Yes, C/C++	Yes, Objective-C	No	Yes, C++	Yes, C++, C#, VB.NET
<b>Digital Certificates</b>	Required for distribution	Required for distribution	Required for distribution	Required for distribution	No, but may be required to access certain APIs	Supported but can be disabled	Supported, required for some phones
<b>Retail Support</b>	Retail support via Android Market; \$25/yr.; No revenue share to 30% revenue to carrier	Retail support via Blackberry App World; \$200 for 10 application submissions; 20% revenue to RIM; free apps ok	To Be Announced	Full retail support via App Store; \$99/year; 30% revenue to Apple; free apps ok	Varies. Options include Carrier Deck and third party aggregators such as getJar. For Nokia Devices Ovi Store	Retail support via Ovi Store.	Retail support via Windows Marketplace for Mobile; \$99yr for 5 submissions \$99 for each additional submission in a year; 30% revenue to Microsoft
<b>Platform Maturity</b>	New	Moderate	Mature	Moderate	Mature	Mature	Mature
<b>Community Support</b>	Excellent	Excellent	Moderate	Excellent	Moderate	Excellent	Excellent
<b>App Installation Method</b>	Direct via Android Market	RIM Store	To Be Announced	Direct via iTunes App Store		Direct, also via PC tools	Direct via ActiveSync. (Microsoft may move to an app store model)

Category	Android	BlackBerry	BREW MP	iPhone	J2ME	Symbian S60	Windows Phone 7
<b>Emulator Available</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes
<b>Remote Debugging</b>	Yes	Yes	Yes	Yes	No	Yes	Yes
<b>Touchscreen Support</b>	Single touch. (Limited multi-touch support)	Single-touch	Single-touch	Multi-touch	Single-touch	Single-touch (5 <sup>th</sup> Edition only)	Single-touch
<b>Underlying Architecture</b>	Linux	Blackberry	BREW	Cocoa Touch / Mac OS X	Varies	Symbian	Windows
<b>Flash Support</b>	No	Support coming in 2010	Yes	No	Varies	Yes	Yes
<b>Java Support</b>	Yes	Yes	Yes	No	Yes	Yes	Yes

Table 2 — Android SDK compared to other popular mobile SDKs. Source: Adapted from Engadget.

## 1.5 Terms and Acronyms

Acronym	Definition
ADT	Android Development Tools
API	Application Programming Interface
IDE	Integrated Development Environment
DHCP	Dynamic Host Configuration Protocol
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
J2ME	Java 2 Micro Edition
LWUIT	Light Weight User Interface Toolkit
MIME	Multipurpose Internet Mail Extensions
MP3	Moving Picture Experts Group Layer-3 Audio
NDK	Native Development Kit
OHA	Open Handset Alliance
RAM	Random-access Memory
SDK	Software Development Kit
SMS	Short Messaging Service
URI	Universal Resource Identifier
URL	Universal Resource Locator

Acronym	Definition
UMTS	Universal Mobile Telecommunication System
VM	Virtual Machine
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Table 3 — Terms and Acronyms

## 1.6 Why Develop for Android

The Android platform differs in some technical and business aspects from its competitors, which may be of interest to application developers.

### 1.6.1 Technical Advantages

Unlike many competitors, Android is built upon an open-source platform, and most of the Android code is released under the free software/open source Apache License. For developers who want to write very specialized applications, particularly applications which depend on functionality for which there are no libraries, this is a major advantage over other systems. Android applications are written in the Java programming language, which is a powerful, mature and very widely adopted language in the global development community. Additionally, the educational resources available for learning Java far outnumber those for other languages like the iPhone's Objective-C, and many find the Java syntax easy to learn. Android's Java is not exactly the same as J2ME, however, the vast majority of existing J2ME applications can be ported to Android with minor modifications.

*NOTE: While Google releases most of the Android code to the community, Google typically only does so once it launches a new Android version (i.e. Android 2.0), as opposed to making code available along the way as the code is checked in.*

## 1.6.2 Business Advantages

Google's venue for selling applications — Android Market — is much more of an open marketplace than Apple's iTunes App Store. Where Apple strives to ensure applications meet its defined standards of quality, utility and appropriateness, Google simply checks that applications meet the terms developers agree to when they sign up for the Android Market. Applications are not evaluated on any other grounds, so developers will likely have more freedom to create the type of application or content they like.

## 2. Developing Applications

Developers will use the tools and APIs in the Android SDK to develop Android applications. Android applications are packaged into a bundle which contains the compiled application code, data files and other resources the application needs. Compiled Android programs have the .dex extension (signifying Dalvik Executable files). The .dex files are then zipped into an application bundle — a single file with an .apk extension called an *Android Package*. Application developers can distribute Android Packages and users can install these packages on their Android devices.

### 2.1 How Android Executes Applications

Android uses the Dalvik virtual machine to run applications safely. The Dalvik virtual machine relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Since each application runs its own Linux process in its own VM, its code and data are kept completely isolated from all other applications running on the device. Even with this extra layer of security, applications can still interact with each other, share user data or other resources.

- Applications can share the same user ID, which allows them to see each other's files,
- Applications with the same user ID can also be made to run in the same VM to save system resources, and
- Applications can be written to make their elements available to other applications, and applications can leverage these elements, such as using another app's UI elements like a scrollbar or viewer. The calling application doesn't need to incorporate the code of the host application,

or even link to it. Android handles this automatically, with Application Components.

## 2.2 Application Components

Android applications are written using application components, which allows for sharing of application resources among all applications installed on an Android device. This means an application can ‘publish’ its resources, such as user interface elements, so that they can be used by other applications on the device, and vice versa. Android allows the sharing of components with other applications by automatically instantiating the Java objects responsible for the desired component. This means that applications do not need to subsume or even link to the application components they want to share — they simply ask Android to run the desired component within the application that contains it.

Since the system must be able to start an application process when any part of it is needed, and instantiate the Java objects for that part, Android applications don't have a single entry point for everything in the application (no `main()` function, for example). Rather, they have essential components that the system can instantiate and run as needed.

There are four types of components:

- Activities,
- Services,
- Broadcast Receivers, and
- Content Providers.

### 2.2.1 Activities

An activity presents a visual user interface for an action the user can undertake. For example, an activity might present a list of menu items users can choose from, or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings.

An application user interface is typically comprised of many activities (for various functions and windows) to form a cohesive user experience. However, each activity is independent of the others. Activities are implemented as a subclass of the Activity base class, and given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen.

### 2.2.1.1 Tasks

Related activities can be grouped into a stack called a *Task*. What the user perceives as an application — with its various windows, controls and displays, is actually a series of tasks, including activities from other applications.

Tasks give the illusion that a single application is providing all the functionality, because the user does not see an application switch when moving from one activity to another, and because the user interface is fast and responsive. To understand what is really happening, imagine a stack of cards, each one representing an activity.

At the bottom of the stack is the root activity — the one that began the task in the first place. Typically this is the activity the user selected in the application launcher, such as the Contacts application. The activity at the top of the stack is one that's currently running and currently displaying its user interface. When one activity starts another activity, the new activity becomes the top card on the stack, directly on top of the activity that launched it. When the user finishes with the top activity (such as dismissing a window or pressing the Back key), this activity at the top is ended and is removed from the stack, revealing the previous task, which resumes as the running activity.

For example, imagine the Contacts application — an application which displays the address book on the phone. The user launches the Contacts application, which instantiates a series of activities in a stack. The user chooses a contact to view its details such as the street address of the contact. The current activity is a viewer activity which is rendering the text of the address on the screen. Imagine the application can show a map of the address to the user. Android has built-in activities which can generate maps, so rather than writing the code to do this in the Contacts application, the current activity in Contacts creates an Intent object with the required information and passes it to `startActivity()`. The map viewer

will display the map in the top-most activity of the task stack. When the user hits the BACK key, the map disappears and the address view of the contact reappears on the screen.

To the user, the entire activity appeared as though it was all done within the same application; however, the map rendering functionality was defined in another application and ran in a separate process, although both activities appeared in the same task.

A task and all the activities it contains move together in a unit when the task is sent to the background and another task is moved to the foreground, for example when the user returns to the Home screen to open another application. The important distinction is that the activity “cards” within a task are never interleaved with other task stacks — they move as a unit, like two separate decks of cards. Therefore, the Back button will always move between activity cards in the current stack, not move between activity cards of all tasks chronologically in the way a web browser’s back button moves backwards chronologically through a history of pages.

### 2.2.2 Services

A service is an application with no visual user interface, which runs in the background. Services can perform tasks for other applications, for example, a service might play background music as the user runs other applications. Services might also be used to fetch data over the network, process data and return the result to an activity that needs it. Each service extends the Service base class.

Some applications might be written to use background services in addition to foreground activities. For example, consider a media player. It may consist of activities to generate an album list, display cover art, and provide playback controls. When the user begins playback of a chosen song, the activity will request a service to handle MP3 playback. Since the service can run in the background regardless of what’s happening in the foreground, the music can play without interruption even if the user is active in other applications such as the web browser or email client.

## LEVERAGING SERVICES

Each service class must have a corresponding `<service>` declaration in its package's `AndroidManifest.xml`. Services can be started with `Context.startService()` and `Context.bindService()`.

Services run in the main thread of the process that spawns them, so if the service will be doing CPU intensive tasks (like MP3 playback), the service should spawn a separate thread for those tasks to avoid blocking the application.

### 2.2.3 Broadcast Receivers

Broadcast receivers are components which are always running, listening for events and carrying out some action in response. Broadcast receivers can't display anything on the device, but they can start activities which can generate a user interface, or use `NotificationManager` to display an alert to the user. For example, a broadcast receiver may be listening for status messages from the power manager, and will launch `NotificationManager` to display a low battery warning when the battery charge is below a certain level.

An application can have any number of broadcast receivers to respond to any announcements it considers important. All receivers extend the `BroadcastReceiver` base class.

In addition, applications can initiate broadcasts. For example, an application can let other applications know that some data has been downloaded to the device and is available for them to use.

## IMPLEMENTING A BROADCAST RECEIVER

The `<receiver>` call declares a broadcast receiver as one of the application's components, enabling the application to receive intents from the system or other applications even when other components of the application are not running.

Developers can make a broadcast receiver known to the system by either:

- Declaring it in the manifest file with the `<receiver>` element, or
- Create the receiver dynamically in code and register it with the `Context.registerReceiver()` method

```
<receiver android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

*Coding Example: Making broadcast receiver known to the system.*

## 2.2.4 Content Providers

Content providers are components which store and retrieve data so that it is accessible to other applications. In Android, there is no data storage area common to all applications. Sharing data between applications can only be accomplished using content providers, which ensures data privacy and security.

Applications can store their data privately, within their resource bundle, or publically in a content provider. Android ships with a number of content providers for example, the contacts data is used by multiple applications and is therefore stored in a content provider rather than within the address book application bundle.

### ACCESSING A CONTENT PROVIDER

Applications can, with proper permissions, query content providers to find out what content they contain, and whether they can read and modify this information..

When requests are made via the `ContentResolver`'s methods call, the system inspects the authority of the given URI and passes the request to the content provider registered with the authority. How the rest of the URI is interpreted is up to the content provider.

The primary methods that need to be implemented for the `ContentResolver` are:

- `query(Uri, String[], String, String[], String)` which returns data to the caller,

- `insert(Uri, ContentValues)` which inserts new data into the content provider,
- `update(Uri, ContentValues, String, String[])` which updates existing data in the content provider,
- `delete(Uri, String, String[])` which deletes data from the content provider, and
- `getType(Uri)` which returns the MIME type of data in the content provider.

A query may also be made via the `Activity.managedQuery()` method, which causes the Activity to manage the lifecycle of the Cursor. In the following example, Android accesses the contact database on the device from within an Activity and retrieves record #2:

```
// Create URI for contact with an ID of 2
contactUri = ContentUris.withAppendId(Contacts.People.CONTENT_URI,
2);
```

*Coding Example: Accessing the contact database on the device from within an activity.*

```
// Request a specific record.
Cursor managedCursor = managedQuery(
    contactUri,
    ContentUris.withAppendedId(Contacts.People.CONTENT_URI, 2),
    projection,    // Which columns to return.
    null,          // WHERE clause.
    null,          // WHERE clause value substitution
    People.NAME + " ASC"); // Sort order.
```

*Coding Example: Retrieving a record*

## CREATING A CONTENT PROVIDER

An application can make its data public in one of the following ways:

- By creating a `<ContentProvider>` subclass, or
- By adding data to an existing content provider if one exists which controls the same type of data to be posted, and if the application has permission to write to it.

A content provider can be created by:

- Setting up a system for storing the data, either using Android's file storage methods, SQLite databases, or a custom method you write. The `SQLiteOpenHelper` and `SQLiteDatabase` classes are available to assist with creating databases and managing them,
- Extending the `ContentProvider` class to provide access to the data, or
- Declaring the content provider in the manifest file for your application (`AndroidManifest.xml`)

Whenever there's a request that should be handled by a particular component, Android will automatically start the component if needed, or will even create the instance of a component if necessary.

## 2.3 The Android Manifest File (AndroidManifest.xml)

Every application must have an `AndroidManifest.xml` file in its root directory. The manifest presents essential information about the application to the Android operating system, which Android requires before it will run any of the application's code. If you use the Eclipse development environment with the Android Development Tools plug-in, the manifest file is created for you automatically.

In addition, `AndroidManifest.xml` does the following:

- Names the Java package for the application, which is the application's unique identifier,
- Describes the components of the application (activities, services, broadcast receivers, and content providers) which let Android know what the components are and under what conditions they can be launched,
- Determines which processes will host application components,
- Declares which permissions the application must have and permissions it requires other applications to have to interact with it,

- Lists the Instrumentation classes that provide profiling and other information as the application is running,
- Declares the minimum level of the Android API that the application requires, and
- Lists the libraries that the application must be linked against.

The example below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is documented in full in a separate file.

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

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
```

```
        <meta-data />
    </receiver>

    <provider>
        <grant-uri-permission />
        <path-permission />
        <meta-data />
    </provider>

    <uses-library />

</application>

</manifest>
```

*Coding Example: General structure of the manifest file and every element that it can contain*

## 2.4 Activating Application Components Using Intents

The three of the four core application components we have discussed — activities, services, and broadcast receivers — are activated through asynchronous messages, called *intents*. An Intent is a message which requests application components to activate to perform a task, such as requesting that the phone dial a number passed to it from the web browser.

An `Intent` object holds the content of the message. For activities and services, the intent names the action being requested and specifies the URI of the data to act on, among other things. For example, an intent might request an activity to present an image to the user or let the user edit some text. For broadcast receivers, the `Intent` object names the action being announced. For example, it might announce to listening receivers that the camera button has been pressed.

There are separate mechanisms for delivering intents to each type of component:

- An Intent object is passed to `Context.startActivity()` or `Activity.startActivityForResult()` to launch an activity or get an existing activity to do something new. It can also be passed to `Activity.setResult()` to return information to the activity that called `startActivityForResult()`,

- An Intent object is passed to `Context.startService()` to initiate a service or deliver new instructions to an ongoing service. Similarly, an intent can be passed to `Context.bindService()` to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running, or
- Intent objects passed to any of the broadcast methods (such as `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, or `Context.sendStickyBroadcast()`) are delivered to all interested broadcast receivers. Many kinds of broadcasts originate in system code.

In each case, Android finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to `startActivity()` is delivered only to an activity, never to a service or broadcast receiver, and so on.

```
Uri = Uri.parse("geo: 32.779523,-96.798918");
Intent myIntent = new Intent(android.content.Intent.ACTION_VIEW,
    locationUri);
startActivity(myIntent);
```

*Coding Example: Intent response*

## 2.4.1 Structure of Intent Objects

An Intent object contains information about the target component to carry out the action, the requested action, and the data to be used as an input for the action, among others.

### COMPONENT NAME

The intent object can request a target component by name, or it can leave it up to Android to determine a suitable target based on the context and content of the intent call, much in the same way a desktop operating system will launch a suitable application to open a document if no specific application is requested.

The component name is set by `setComponent()`, `setClass()`, or `setClassName()` and read by `getComponent()`

## ACTION

`Action` is a string naming the action to be performed. The `action` string can also be used by broadcast intents to report to listeners an action that just took place. Below are constants supported by the `action` string by default, although developers can create their own action strings for their applications.

Constant	Target Component	Action
<code>ACTION_CALL</code>	activity	Initiate a phone call
<code>ACTION_EDIT</code>	activity	Display data for the user to edit
<code>ACTION_MAIN</code>	activity	Start up as the initial activity of a task, with no data input and no returned output
<code>ACTION_SYNC</code>	activity	Synchronize data on a server with data on the mobile device
<code>ACTION_BATTERY_LOW</code>	broadcast receiver	A warning that the battery is low
<code>ACTION_HEADSET_PLUG</code>	broadcast receiver	Event when headset has been plugged into or removed from the device
<code>ACTION_SCREEN_ON</code>	broadcast receiver	The screen has been turned on
<code>ACTION_TIMEZONE_CHANGED</code>	broadcast receiver	The setting for the time zone has changed

Table 4 — Default action strings supported by Android.

## DATA

Data contains the URI of the data to be acted on and the MIME type of that data. Different actions are paired with different kinds of data specifications. For example, if the action field is `ACTION_EDIT`, the data field would contain the URI of the document to be displayed for editing. If the action is `ACTION_CALL`, the data field would be a `tel:` URI with the number to call. Similarly, if the action is `ACTION_VIEW` and the data field is an `http:` URI, the receiving activity would be called upon to download and display whatever data the URI refers to.

When matching an intent to a component that is capable of handling the data, it's important to know the type of data (its MIME type) in addition to its URI. For example, a component able to display image data should not be called upon to play an audio file.

### CATEGORY

Category is a string containing information about the kind of component that should handle the intent. Some categories are built-in (below), and new categories can be created and placed in an Intent object as needed.

Constant	Description
<code>CATEGORY_BROWSABLE</code>	The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message
<code>CATEGORY_GADGET</code>	The activity can be embedded inside of another activity that hosts gadgets
<code>CATEGORY_HOME</code>	The activity displays the home screen, the first screen the user sees when the device is turned on or when the HOME key is pressed
<code>CATEGORY_LAUNCHER</code>	The activity can be the initial activity of a task and is listed in the top-level application launcher
<code>CATEGORY_PREFERENCE</code>	The target activity is a preference panel

*Table 5 — Default intent categories supported by Android.*

### EXTRAS

Extras allow key-value pairs for additional information that should be delivered to the component handling the intent. Extras are useful for handling special information specific to the action, for example a `SHOW_COLOR` action (which is a custom action) would need a color value set in an extra key-value pair.

### FLAGS

Flags instruct Android how to launch an activity (for example, which task the activity should belong to) and how to treat the activity after it's launched (for example, whether it belongs in the list of recent activities). All flags are defined in the Intent class.

### 3. Developing App Widgets

App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates. These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider. An application component that is able to hold other App Widgets is called an App Widget host.



Figure 2 — Some examples of App Widgets running on the Android home screen. Courtesy Google.

#### 3.1 Creating App Widgets

To create an App Widget, you need the following:

1. **AppWidgetProviderInfo** object

Describes the metadata for an App Widget, such as the App Widget's layout, update frequency, and the `AppWidgetProvider` class. This should be defined in XML,

**2. `AppWidgetProvider` class implementation**

Defines the basic methods that allow you to programmatically interface with the App Widget, based on broadcast events. Through it, you will receive broadcasts when the App Widget is updated, enabled, disabled and deleted, and

**3. View layout**

Defines the initial layout for the App Widget, defined in XML.

*TIP: Additionally, App Widget configuration Activities are available, which launch a configuration activity for your application when a user adds it to the device Home page. This allows an App Widget to be configured by the user when it is created.*

## 3.2 Layouts and Views

App Widgets are placed inside a container defined in the `View` class. Containers are defined using XML and placed in the project's `res/layout/` directory.

See section 5.1 below on Layouts for more information about creating containers. Note that App Widget layouts are based on `RemoteViews`, which do not support every kind of layout or view widget

## 3.3 Handling Events

The `AppWidgetProvider` class extends `BroadcastReceiver` as a convenience class to handle App Widget broadcasts.

The `AppWidgetProvider` receives event broadcasts that are relevant to the App Widget, such as when the App Widget is updated, deleted, enabled, or disabled. When these broadcast events occur, the `AppWidgetProvider` receives the following method call:

```
onUpdate (Context, AppWidgetManager, int[])
```

*Coding Example: Broadcast event relevant to the app widget*

This is called to update the App Widget at intervals defined by the `updatePeriodMillis` attribute in the `AppWidgetProviderInfo`

### ADDITIONAL APP WIDGET EVENTS

- **onDeleted(Context, int[])**  
Called when an App Widget is deleted from the App Widget host.
- **onEnabled(Context)**  
Called when an instance the App Widget is created for the first time. (i.e. not called if the user adds a subsequent instance of the App Widget).
- **onDisabled(Context)**  
Called when the last instance of the App Widget is deleted from the App Widget host. This can be a flag to perform clean-up tasks, such as deleting a temporary database.
- **onReceive(Context, Intent)**  
Called for every broadcast, and before each of the above callback methods.

**TIP:** *If the device is asleep when it is time for an update (as defined by `updatePeriodMillis`), then the device will wake up in order to perform the update. If you don't update more than once per hour, this probably won't cause significant problems for the battery life. If, however, you need to update more frequently and/or you do not need to update while the device is asleep, then you can instead perform updates based on an alarm that will not wake the device. To do so, set an alarm with an Intent that your AppWidgetProvider receives, using the AlarmManager. Set the alarm type to either `ELAPSED_REALTIME` or `RTC`, which will only deliver the alarm when the device is awake. Then set `updatePeriodMillis` to zero ("0").*

**TIP:** The most important `AppWidgetProvider` callback is `onUpdated()` because it is called when each App Widget is added to a host (unless you use a configuration Activity). If your App Widget accepts any user interaction events, then you need to register the event handlers in this callback. If your App Widget doesn't create temporary files or databases, or perform other work that requires clean-up, then `onUpdated()` may be the only callback method you need to define.

## 4. Communications

Android supports communications over Wi-Fi, Bluetooth and of course the mobile data network. Network IP communications are possible over Wi-Fi and the mobile data network. Android supports two APIs to handle network communications: Android.net and Java.net APIs.

Android subsystems handle network acquisition and management so that you can abstract your application from dealing with the details, however, there are a variety of methods for querying about network status and specifying particulars if your application requires fine-grain control.

### 4.1 Basic Networking

Basic networking in Android is similar to standard Java networking since Android supports the Java.net APIs to handle tasks such as streaming, managing datagram sockets, Internet addresses and HTTP requests.

The following standard Java.net interfaces are supported on Android:

Interface	Description
<code>ContentHandlerFactory</code>	Defines a factory which is responsible for creating a <code>ContentHandler</code> .
<code>DatagramSocketImplFactory</code>	This interface defines a factory for datagram socket implementations.
<code>FileNameMap</code>	Defines a scheme for mapping a filename type to a MIME content type.
<code>SocketImplFactory</code>	This interface defines a factory for socket implementations.
<code>SocketOptions</code>	Defines an interface for socket implementations to get and set socket options.
<code>URLStreamHandlerFactory</code>	Defines a factory which creates an <code>URLStreamHandler</code> for a specified protocol.

Table 6 — Networking interfaces in Android

In addition, the Android.net API adds these classes:

Class	Description
<code>ConnectivityManager</code>	Answers queries about the state of network connectivity
<code>Credentials</code>	Handles UNIX credentials on UNIX domain sockets
<code>DhcpInfo</code>	Retrieves the results of a DHCP request
<code>LocalServerSocket</code>	Creates inbound UNIX-domain sockets
<code>LocalSocket</code>	Creates a (non-server) socket in the UNIX-domain namespace
<code>LocalSocketAddress</code>	A UNIX-domain (AF_LOCAL) socket address
<code>MailTo</code>	Parses mailto scheme URLs. Can be queried for the parsed parameters
<code>NetworkInfo</code>	Describes the status of a network interface of a given type (either mobile or Wi-Fi).
<code>Proxy</code>	Provides access to the user and default proxy settings
<code>SSLCertificateSocketFactory</code>	Allows SSL certificate chain validation to be skipped when negotiating an SSL session.
<code>Uri</code>	Immutable URI reference.
<code>Uri.Builder</code>	Helper class for building or manipulating URI references.
<code>UrlQuerySanitizer</code>	Sanitizes the Query portion of a URL.
<code>UrlQuerySanitizer.IllegalCharacterValueSanitizer</code>	Sanitize values based on which characters they contain.
<code>UrlQuerySanitizer.ParameterValuePair</code>	A simple tuple that holds parameter-value pairs.

Table 7 — Android networking APIs

## 4.2 Connections and Management

The Android.net API offers the `ConnectivityManager` class to set up and manage connections.

`ConnectivityManager` can be queried for the state of network connectivity, and it can also notify applications when network conditions change.

`ConnectivityManager` does the following:

- Monitors network connections (Wi-Fi, GPRS, UMTS, etc.),
- Sends broadcast intents when network connectivity changes,
- Attempts to "fail over" to another network when current network connectivity, and
- Provides an API that allows applications to query the coarse-grained or fine-grained state of the available networks

### 4.2.1 Setting Up a Network Connection

Setting up a network connection or requesting data is a simple procedure, and Android abstracts many of the details. Your application will call the public method `requestRouteToHost (int networkType, int hostAddress)`

This instructs Android to ensure a network route exists to deliver traffic to the specified host via the specified network interface. Android will return `true` on success; `false` on failure. If a request is made to set up a connection which already exists, Android will ignore the request, but will treat it as successful.

Parameter	Description
<code>networkType</code>	The type of the network requested for routing data to specified host (i.e. mobile or Wi-Fi)
<code>hostAddress</code>	The IP address of the target host

Table 8 — Parameters for setting up a network connection

**TIP:** The `getBackgroundDataSetting ()` call will tell your application whether fetching of data in the background is permitted. If the result is `<false>`, applications should not use the network if the application is not in the foreground. Developers should respect this setting, and check the value of this before performing any background data operations.

If your application can benefit from accessing the network passively when it is in the background, set your application to listen to `ACTION_BACKGROUND_DATA_SETTING_CHANGED` to be notified if the user changes settings allowing this operation.

## 4.2.2 Choosing the Network Bearer

Android will automatically use the default network for all requests, although your application may request a specific network type for its data requests. This can be useful if, for example, if your application downloads large files. In this case you may wish to only use Wi-Fi connections and not mobile connections.

### DETERMINING CURRENT NETWORK TYPE

Your application can query to see which network type is currently in use (i.e. mobile or Wi-Fi) by calling:

- `getType ()`

### SPECIFYING A NETWORK TYPE

Your application can specify which network type to activate by using:

- `public static final int TYPE_MOBILE, or`
- `public static final int TYPE_WIFI`

When the desired connection type as specified above is active, all data traffic from your application will use this connection type by default.

### 4.2.3 Connection Maintenance

Android can provide status about the current state of network connectivity through the `android.net.NetworkInfo` class. This class provides functions for learning network state either in detail, or simplified terms (either 'Connecting', 'Connected', 'Disconnecting', 'Disconnected', 'Suspended' or 'Unknown').

- enum `NetworkInfo.DetailedState` provides fine-grained network state information,
- enum `NetworkInfo.State` provides coarse-grained network state information.

#### GETTING INFORMATION ABOUT NETWORK CONNECTIONS

Other calls in the `NetworkInfo` class are useful for general status, or for troubleshooting:

Public Method	Description
<code>NetworkInfo.DetailedState /getDetailedState()</code>	Reports the current fine-grained state of the network
<code>getExtraInfo()</code>	Report the extra information about the network state, if any was provided by the lower networking layers
<code>getReason()</code>	Report the reason an attempt to establish connectivity failed, if one is available.
<code>NetworkInfo.State/getState()</code>	Reports the current coarse-grained state of the network.
<code>getSubtype()</code>	Return a network-type-specific integer describing the subtype of the network.
<code>getSubtypeName()</code>	Return a human-readable name describing the subtype of the network.
<code>getType()</code>	Reports the type of network currently in use (i.e. mobile or Wi-Fi)
<code>getTypeName()</code>	Return a human-readable name describe the type of the network, for example "WIFI" or "MOBILE".
<code>isAvailable()</code>	Indicates whether network connectivity is possible.
<code>isConnected()</code>	Indicates whether network connectivity exists and it is possible to establish connections and pass data.

Public Method	Description
<code>isConnectedOrConnecting()</code>	Indicates whether network connectivity exists or is in the process of being established.
<code>isFailover()</code>	Indicates whether the current attempt to connect to the network resulted from the <code>ConnectivityManager</code> trying to fail over to this network following a disconnect from another network.
<code>isRoaming()</code>	Indicates whether the device is currently roaming on this network.
<code>toString()</code>	Returns a string containing a concise, human-readable description of this object.

Table 9 — Public methods for obtaining network information

## BEING NOTIFIED OF NETWORK CHANGES

The `ConnectivityManager` class, part of `Android.net` provides a number of methods for your application to receive notification of changing network conditions.

Public Method	Description
<code>CONNECTIVITY_ACTION</code>	This can notify your application that a change in the network condition has been detected; either network has been established or lost. Your application can also query to see if the reason for the change is due to a failover condition
<code>EXTRA_IS_FAILOVER</code>	This indicates whether a connection event is for a given network is an attempt to repair a failover condition.
<code>EXTRA_NO_CONNECTIVITY</code>	Indicates whether there is a complete lack of connectivity
<code>EXTRA_OTHER_NETWORK_INFO</code>	Information is supplied when there is another network that may possibly accept connections
<code>EXTRA_REASON</code>	Indicates why an attempt to connect to a network failed.

Table 10 — Public methods for network event notification

***TIP:** If the connectivity manager is attempting to connect (or has already connected) to another network as a result of a failover condition, the `NetworkInfo` for the new network is also passed as an extra. This lets any receivers of the broadcast know that Android is attempting to repair network connectivity and that applications should not necessarily tell the user that no data traffic will be possible. Instead, this notification can be used to tell your application to wait for another broadcast indicating either that the failover attempt succeeded or failed, and take action accordingly.*

### 4.3 Telephony

As you would expect, Android provides a robust set of APIs for monitoring telephone operations, such as the network type and connection state, plus utilities for manipulating phone number strings.

The classes include:

Public Method	Description
<code>CellLocation</code>	Abstract class that represents the location of the device
<code>NeighboringCellInfo</code>	Represents the neighboring cell information, including Received Signal Strength and Cell ID location
<code>PhoneNumberFormattingTextWatcher</code>	Watches for the user entering phone numbers, and will format them using <code>formatNumber</code>
<code>PhoneNumberUtils</code>	Various utilities for dealing with phone number strings
<code>PhoneStateListener</code>	A listener class for monitoring changes in specific telephony states on the device, including service state, signal strength, message waiting indicator (voicemail), etc.
<code>ServiceState</code>	Contains phone state and service related information
<code>SignalStrength</code>	Contains phone signal strength related information
<code>SmsManager</code>	Manages SMS operations such as sending data, text, and SMS messages

Public Method	Description
<code>TelephonyManager</code>	Provides access to information about the telephony services on the device

Table 11 — Android telephony methods

### 4.3.1 Phone Calls

Android provided a built-in application called Dialer which links user input and application intents with the underlying telephony code, and allows users or applications, respectively, to make phone calls. Applications must have permission (as listed in the Android manifest file — `<uses-permission id="android.permission.CALL_PHONE" />`) to use this function.

```
Intent dialIntent = new Intent(Intent.ACTION_CALL, Uri
    .parse("tel:6365551212"));
startActivity(dialIntent);
```

*Coding Example: Dialer App*

In the above example, Android will call the entered phone number, provided the number is formatted as a valid (IETF RFC 3966) phone number, such as:

- tel:2125551212, or
- tel: (212) 555 1212.

**TIP:** The Dialer can normalize some kinds of schemes, so the formats above are not strictly required, however you should ensure that your formatting scheme will be accepted by the dialer. Use `PhoneNumberUtils` or `Uri.fromParts` factory to generate properly formatted numbers

A similar function action (`ACTION_DIAL`) will pre-dial the number, but will not instruct Android to place the call.

## PHONE NUMBER UTILITIES

Android also provides a group of tools under the `PhoneNumberUtils` class for working with phone numbers, such as formatting numbers for a variety of global locales, comparing phone numbers against the address book for the purposes of matching strings for Caller ID, converting text (i.e. 1-800-FLOWERS) into phone numbers, and so on.

### 4.3.2 SMS

Android supports the creation and sending of plain text messages, multi-part text messages, and data messages. Android also provides a function to automatically divide text messages into multi-part messages.

## IMPLEMENTING TEXT MESSAGE FUNCTIONALITY

Android provides APIs to integrate with its built-in SMS client, which is a simple way to handle text messaging tasks by invoking the built-in SMS client from within your application. To do this, use an Intent object to invoke the built-in SMS application:

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
    sendIntent.putExtra("sms_body", "Content of the SMS goes
here...");
    sendIntent.setType("vnd.android-dir/mms-sms");
    startActivity(sendIntent);
```

*Coding Example: Using an Intent object to invoke built-in SMS app.*

### WRITING YOUR OWN SMS CLIENT

You can also write your own SMS client fairly simply using the SMS classes in the `android.telephony` package.

- Modify the `AndroidManifest.xml` file to grant permission for your application to send and receive SMS messages. You will need to add the permissions `SEND_SMS` and `RECEIVE_SMS`
- Create a layout in the in the `main.xml` file located in the `res/layout` folder to give your application a user interface:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Enter the phone number of recipient"
        />
    <EditText
        android:id="@+id/txtPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Message"
        />
    <EditText
        android:id="@+id/txtMessage"
        android:layout_width="fill_parent"
        android:layout_height="150px"
        android:gravity="top"
        />
    <Button
        android:id="@+id/btnSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send SMS"
        />
</LinearLayout>
```

Coding Example: Creating the UI for Figure 3

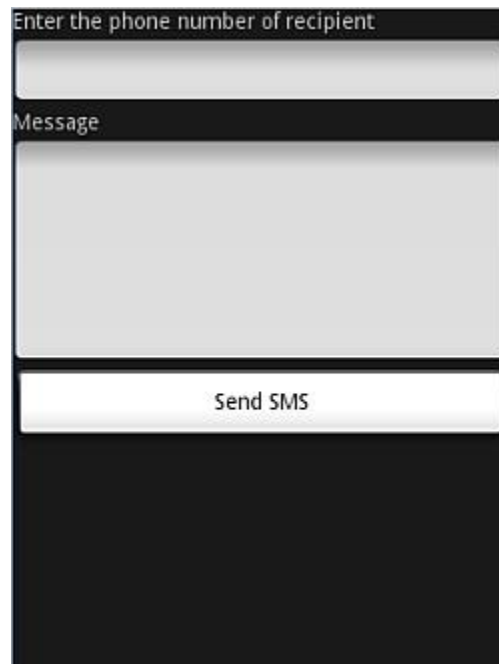


Figure 1 — Android Platform Architecture. Source: Google.

*Note: The button view in Figure 3 is 'wired up' in code so that the UI elements trigger the appropriate functions, such as the `sendSMS ()` call*

- To send an SMS message, use the `SmsManager` class. The main function, the `sendSMS ()` function is defined as follows:

```
public class SMS extends Activity
{
    //...

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        //...
    }

    //---sends an SMS message to another device---
    private void sendSMS(String phoneNumber, String message)
    {
        PendingIntent pi = PendingIntent.getActivity(this, 0,
            new Intent(this, SMS.class), 0);
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, pi, null);
    }
}
```

**Coding Example: Sending an SMS Message**

**NOTE:** Unlike other classes, *SmsManager* class is not directly instantiated; rather it is called through an *SmsManager* object. The *sendTextMessage()* method sends the SMS message with a *PendingIntent*, which will allow you to display another activity, such as a function to monitor the sending of the message.

Parameters	Description
<code>destinationAddress</code>	The address to send the message to
<code>scAddress</code>	The service center address or null to use the current default SMSC
<code>text</code>	The body of the message to send
<code>sentIntent</code>	<p>if not NULL this PendingIntent is broadcast when the message is successfully sent, or failed. The result code will be <code>Activity.RESULT_OK</code> for success, or one of these errors:</p> <ul style="list-style-type: none"> <li><code>RESULT_ERROR_GENERIC_FAILURE</code></li> <li><code>RESULT_ERROR_RADIO_OFF</code></li> <li><code>RESULT_ERROR_NULL_PDU</code></li> </ul> <p>For <code>RESULT_ERROR_GENERIC_FAILURE</code> the <code>sentIntent</code> may include the extra "errorCode" containing a radio technology specific value, generally only useful for troubleshooting.</p> <p>The per-application based SMS control checks <code>sentIntent</code>. If <code>sentIntent</code> is NULL the caller will be checked against all unknown applications, which cause smaller number of SMS to be sent in checking period.</p>
<code>deliveryIntent</code>	This PendingIntent is broadcast when the message is delivered to the recipient.

Table 12 — Parameters for generating SMS messages

### IMPLEMENTING MULTI-PART TEXT MESSAGES

Sending a multi-part text message is essentially the same as a standard text message, with the addition of the `ArrayList<String> parts` parameter.

```
public void sendMultipartTextMessage (String destinationAddress,
String scAddress, ArrayList<String> parts, ArrayList<PendingIntent>
sentIntents, ArrayList<PendingIntent> deliveryIntents)
```

Android can also divide the text message into a multi-part message for you by using the following:

```
public ArrayList<String> divideMessage (String text)
```

*Coding Example: Sending a multi-part message*

## 5. Android UI

In an Android application, the user interface is built using `View` objects within the `View` class and `ViewGroup` objects. `View` objects are user interface elements, which are contained within a spatial layout, the `ViewGroup`. The `ViewGroup` is a special view which can contain other view objects, called Children. There are many types of views and view groups, each of which is a descendant of the `View` class.

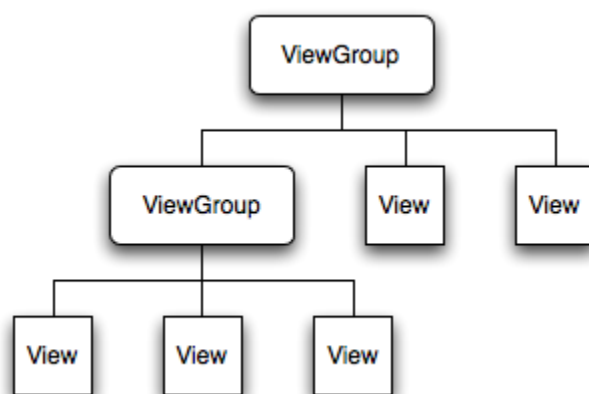


Figure 3 —The Android view hierarchy.

### 5.1 Layouts

Objects are arranged in a spatial container called Layouts, which hold all of the elements that appear to the user. Different types of layouts are available in the “layouts” subclasses, such as linear layouts, tabular layouts and relative layouts

Layouts can be declared in two ways:

- **Declare UI elements in XML**  
Android provides a straightforward XML vocabulary that corresponds to the `View` classes and subclasses, such as those for widgets and layouts, or
- **Instantiate layout elements at runtime**  
Your application can create `View` and `ViewGroup` objects (and manipulate their properties) programmatically.

***TIP:** Developers can use either XML, runtime or both methods for declaring and managing application user interfaces. For example, an application's default layouts could be declared in XML (as the screen appears to the user at application launch). Programmatic changes to the user interface layout can be managed through code which is executed at runtime, allowing the user interface to change based on activities in the application.*

*Note that defining layouts in XML has the advantage of keeping the visual presentation layer separate from application code, which can make it easier to troubleshoot applications, and adapt the visual presentation without changing application code or recompiling.*

### 5.1.1 Layout Types

Android supports common layout types. Layouts are subclasses of `ViewGroup`.

#### FRAME LAYOUT

`FrameLayout` is the simplest type of layout object. A `FrameLayout` is blank rectangular container on the screen which can be filled with a single object, such as a picture. `FrameLayout` objects are simple, and do not allow any control over the position of `Child` objects placed within them. `Child` objects are pinned to the upper left corner of the screen.

Additional `Child` view objects added to the `FrameLayout` will be drawn over previous ones, completely or partially obscuring layers below, depending on the size and transparency of the child overlaying the other layers.

#### LINEAR LAYOUT

A `LinearLayout` aligns all children in a single direction — vertically or horizontally. A vertical `LinearLayout` stacks all children such that one column is created with one child per row. A horizontal `LinearLayout` will line all children along the horizon, making one row, the height of the tallest child, plus padding.

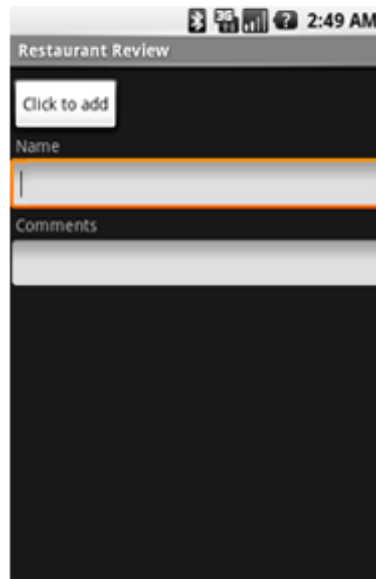


Figure 4 — A `LinearLayout` set up in the vertical orientation. All three children above are 'stacked' one atop the other in a single column.

### TABLE LAYOUT

`TableLayout` positions children into rows and columns (without visible border lines for rows, columns, or cells). The table will have as many columns as the row with the most cells. A table can leave cells empty, but cells cannot span columns, as they can in HTML.

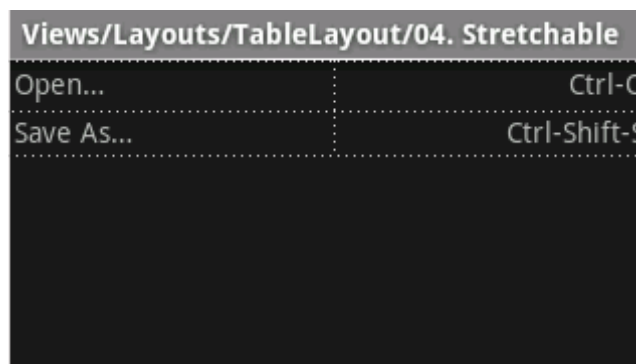


Figure 5 — A Table layout

## RELATIVE LAYOUT

`RelativeLayout` lets child views specify their position relative to the parent view or to each other, for example allowing all items to be aligned along their right borders, or centering items relative to the screen bounds.

In the example below, the text box is aligned relative to the screen center; the OK button aligned relative to the right of the text box; and the Cancel button aligned to the left and top of the OK button.

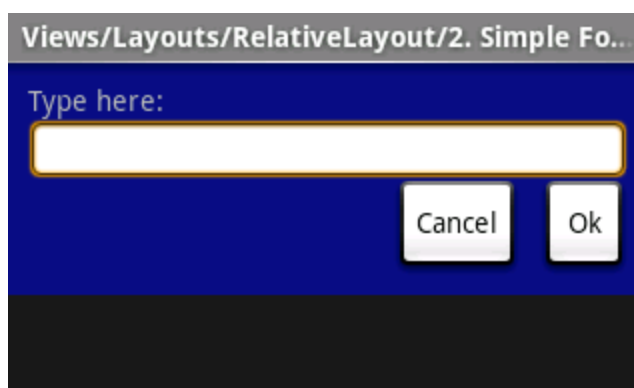


Figure 6 — A Relative layout

### 5.1.2 XML Layout Example

Android's XML vocabulary makes it fast and easy to design UI layouts and the screen elements they contain, using a series of nested elements — similar to the way HTML web pages are created.

Each layout file must contain exactly one root element, which must be a `View` or `ViewGroup` object. Once the root element has been defined, you can add additional layout objects or widgets as child elements to build a View hierarchy that defines the layout.

Below is an XML layout that uses a vertical `LinearLayout` to hold a `TextView` and a `Button`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
  <TextView android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a TextView" />
  <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, I am a Button" />
</LinearLayout>
```

*Coding Example: XML layout utilizing vertical linear layout to hold a text view and button*

**TIP:** You can also draw *View* and *ViewGroups* objects in Java code, using the *addView(View)* methods to dynamically insert new *View* and *ViewGroup* objects.

## 5.2 UI Components

The *View* class contains subclasses called “widgets” (not to be confused with App Widgets). Widgets are fully implemented UI objects, like text fields and buttons. Widgets can be placed inside *ViewGroups* as Children.

The following is a partial list of the UI Widgets available:

Class	Description
<b>AbsSpinner</b>	An abstract base class for spinner widgets.
<b>AnalogClock</b>	This widget display an analogic clock with two hands for hours and minutes.
<b>AutoCompleteTextView</b>	An editable text view that shows completion suggestions automatically while the user is typing.
<b>Button</b>	Button represents a push-button widget.
<b>CheckBox</b>	A checkbox is a specific type of two-states button that can be either checked or unchecked.
<b>CheckedTextView</b>	An extension to TextView that supports the Checkable interface.
<b>Chronometer</b>	Class that implements a simple timer.
<b>CompoundButton</b>	A button with two states, checked and unchecked.
<b>DatePicker</b>	A view for selecting a month / year / day based on a calendar like layout.
<b>DialerFilter</b>	
<b>DigitalClock</b>	Like AnalogClock, but digital.
<b>EditText</b>	EditText is a thin veneer over TextView that configures itself to be editable.
<b>ExpandableListView</b>	A view that shows items in a vertically scrolling two-level list.
<b>ExpandableListView.ExpandableListContextMenuInfo</b>	Extra menu information specific to an ExpandableListView provided to the onCreateContextMenu(ContextMenu, View, ContextMenuInfo) callback when a context menu is brought up for this AdapterView.
<b>Filter</b>	A filter constrains data with a filtering pattern.
<b>Filter.FilterResults</b>	Holds the results of a filtering operation.
<b>FrameLayout</b>	FrameLayout is designed to block out an area on the screen to display a single item.
<b>Gallery</b>	A view that shows items in a center-locked, horizontally scrolling list.
<b>GridView</b>	A view that shows items in two-dimensional scrolling grid.
<b>HorizontalScrollView</b>	Layout container for a view hierarchy that can be scrolled by the user, allowing it to be larger than the physical display.
<b>ImageButton</b>	Displays a button with an image (instead of text) that can

Class	Description
	be pressed or clicked by the user.
<b>ImageView</b>	Displays an arbitrary image, such as an icon.
<b>LinearLayout</b>	A Layout that arranges its children in a single column or a single row.
<b>ListView</b>	A view that shows items in a vertically scrolling list.
<b>MediaController</b>	A view containing controls for a MediaPlayer.
<b>PopupWindow</b>	A popup window that can be used to display an arbitrary view.
<b>ProgressBar</b>	Visual indicator of progress in some operation.
<b>QuickContactBadge</b>	Widget used to show an image with the standard QuickContact badge and on-click behavior.
<b>RadioButton</b>	A radio button is a two-states button that can be either checked or unchecked.
<b>RadioGroup</b>	This class is used to create a multiple-exclusion scope for a set of radio buttons.
<b>RatingBar</b>	A RatingBar is an extension of SeekBar and ProgressBar that shows a rating in stars.
<b>RelativeLayout</b>	A Layout where the positions of the children can be described in relation to each other or to the parent.
<b>RemoteViews</b>	A class that describes a view hierarchy that can be displayed in another process.
<b>Scroller</b>	This class encapsulates scrolling.
<b>ScrollView</b>	Layout container for a view hierarchy that can be scrolled by the user, allowing it to be larger than the physical display.
<b>SeekBar</b>	A SeekBar is an extension of ProgressBar that adds a draggable thumb.
<b>SlidingDrawer</b>	SlidingDrawer hides content out of the screen and allows the user to drag a handle to bring the content on screen.
<b>Spinner</b>	A view that displays one child at a time and lets the user pick among them.
<b>TabHost</b>	Container for a tabbed window view.
<b>TabHost.TabSpec</b>	A tab has a tab indicator, content, and a tag that is used to keep track of it.
<b>TableLayout</b>	A layout that arranges its children into rows and columns.
<b>TabWidget</b>	Displays a list of tab labels representing each page in the parent's tab collection.

Class	Description
<b>TimePicker</b>	A view for selecting the time of day, in either 24 hour or AM/PM mode.
<b>Toast</b>	A toast is a view containing a quick little message for the user. The toast class helps you create and show those.
<b>ToggleButton</b>	Displays checked/unchecked states as a button with a "light" indicator and by default accompanied with the text "ON" or "OFF".
<b>VideoView</b>	Displays a video file.
<b>ViewAnimator</b>	Base class for a FrameLayout container that will perform animations when switching between its views.
<b>ViewFlipper</b>	Simple ViewAnimator that will animate between two or more views that have been added to it.
<b>ViewSwitcher</b>	ViewAnimator that switches between two views, and has a factory from which these views are created.
<b>ZoomControls</b>	The ZoomControls class displays a simple set of controls used for zooming and provides callbacks to register for events.

Table 13 — A partial list of the UI widgets available in Android.

### 5.3 Menus

Android offers an easy programming interface for developers to provide standardized application menus for various situations.

Three types of application menus are available:

#### OPTIONS MENU

This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key. Within the Options Menu are two groups of menu items:

- **Icon Menu**  
The collection of items initially visible at the bottom of the screen when the MENU key is pressed, and
- **Expanded Menu**  
This is a vertical list of items exposed by the "More" menu item from the Icon Menu.

### CONTEXT MENU

This is a floating list of menu items that may appear when the user presses on a View (such as a list item).

### SUBMENU

This is a floating list of menu items that is revealed by an item in the Options Menu or a Context Menu.

## 5.3.1 Menu Item Intents

Menu items can also be used to issue Intents, as a way to directly launch an activity or application. There are two ways to do this:

- Define an Intent and assign it to a single menu item, or
- Define an Intent and allow Android to search the device for matching activities and dynamically show these matches in a list

```
MenuItem menuItem = menu.add(0, PHOTO_PICKER_ID, 0, "Select  
Photo");  
menuItem.setIntent(new Intent(this, PhotoPicker.class));
```

*Coding Example: An Intent for a single item*

## 5.4 Events

Android provides multiple ways to intercept and act on events created as your application is used. The general approach is to capture events from the specific View object that the user interacts with, by using the `View` class.

### EVENT LISTENERS

An event listener is an interface in the `View` class that contains a single callback method. Android will call these methods when the listener detects that a user has interacted with an item in the user interface.

The event listener interface includes the following callback methods:

Method	Description
<b>onClick()</b>	Called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.
<b>onLongClick()</b>	Called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).
<b>onFocusChange()</b>	Called when the user navigates onto or away from the item, using the navigation-keys or trackball.
<b>onKey()</b>	Called when the user is focused on the item and presses or releases a key on the device.
<b>onTouch()</b>	Called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).
<b>onCreateContextMenu()</b>	Called when a Context Menu is being built (as the result of a sustained "long click"). See the discussion on context menus in Creating Menus for more information.

Table 14 — Event listener interface callback methods

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Capture our button from layout
    Button button = (Button) findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

*Coding Example: How to register an “on-click” listener for a button*

**TIP:** *It is also possible to implement an event listener as part of an Activity using `OnClickListener`, which provides the same functionality without loading extra classes and objects.*

## 6. Writing for the Web

Android uses WebKit, a modern, powerful and standards-compliant browser engine. WebKit is an open source project, with contributors from KDE project, Apple, Nokia, Google, RIM, Palm, and others. WebKit is a particularly attractive browser engine for developers because it has a modern, streamlined code base, supports W3C standards, and is open-source so that developers can enhance the codebase. Because WebKit runs on both desktop systems and mobile systems, full-fledged web applications are possible on mobile devices.

### 6.1 Mobile Web/XHTML Sites

Since WebKit is such a capable rendering engine, many sites or web applications developed for desktop browsers may work perfectly well in a mobile WebKit environment, and display with the same fidelity. While this means you do not necessarily need to develop a simplified version of your website or application for technical reasons as was often needed in the past, you may still want to consider tailoring your code to make it more usable and relevant on mobile devices.

#### USABILITY CONSIDERATIONS

When designing applications and sites for mobile devices, it's often a good idea to make pages less information-dense, and make controls larger than typical for desktop browser use. This is because users may be attempting to interact with your site or application on the go, which means they may have less time to read the screen and 'figure out' what to do. Also, consider that your site or application may be viewed on a multi-touch device, requiring users to touch links rather than click on them. Therefore, links should be larger (or appear as buttons), spaced farther apart, and hover styles should not be used.

#### DEVELOPING MOBILE SITES AND WEB APPLICATIONS FOR ANDROID

If your existing site or web application works on iPhone, it will likely work on Android with little modification, and the same development techniques used on iPhone will work on Android as well.

In addition to the usability tips above, functions relating to the mouse may not make sense when viewing a web page or application in a multi-touch mobile browser. For example, events such as `mouse_move()`, `mouse_down()` may fire, but



they may not be returned in the same order expected, or may not be returned at all.

Make use of the `viewport` metatag, which is a special addition for mobile browsers (it is ignored by browsers which do not support it). With `viewport`, developers can specify the initial scaling of the webpage to make it easier to read as it first loads.:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0 user-scalable=yes" />
```

*Coding Example: Initial scaling of the webpage*

While `viewport` and `initial-scale` values are screen independent, meaning that pixel dimensions do not need to be specified, it is always a good idea to know how your web page or application will appear on various target sizes. Also consider that users may view your site in either portrait or landscape orientation, as many mobile browsers support this feature.

### 6.2 Leveraging HTML5

Since version 2.0, WebKit on Android now supports several HTML5 features, which makes even more capable web applications possible. For example, the following HTML5 features are supported:

Feature/Element	Description
<b>Canvas</b>	Allows for dynamic scriptable rendering of bitmap images
<b>Video</b>	Adds support for embedding video in a HTML page, without using a plug-in such as Flash or Silverlight
<b>Geolocation</b>	Provides services based on geographic (GPS) information
<b>App Cache and Database</b>	Stores data locally to allow applications to work offline or to provide protection against lapses in network connectivity
<b>Workers</b>	Allows background computation without blocking the user interface

*Table 15 — HTML5 features supported in Android*

### 6.3 Extending the Browser

WebKit offers a WebView component which can be embedded into applications, allowing them to fully render and use web content.

WebView provides the same rendering fidelity and compatibility as the built-in browser, since it uses the same JavaScript engine and rendering engine the built-in browser uses. WebView provides a rich feature-set, including methods to zoom in and out, perform text searches, navigate through browser history, and more.

***TIP:** WebView is perfect if the goal is to display HTML as a part of an application UI. If much more functionality is needed, launching the built-in browser is probably a better solution.*

The code below creates a window within your application, then loads and renders <http://developer.att.com>. Applications can call JavaScript methods using `webView.loadUrl("javascript:someMethod()")`.

```
WebView webView = new WebView(this);
setContentView(webView);
webView.loadUrl("http://developer.att.com/");
```

*Coding Example: Embedding WebView into an application*

Additionally, JavaScript running in WebView can call out to an Application’s Activity using the `addJavascriptInterface()` method.

**CUSTOMIZING THE WEB VIEW**

WebView allows some degree of customization to suit particular development needs. For example, the following classes, subclasses and methods are available:

Class/Subclass/Method	Description
<b>WebChromeClient</b>	Called when an event which might impact a browser user interface occurs, such as progress updates and JavaScript alerts
<b>WebViewClient</b>	Called when events occur which might impact the rendering of the content, such as errors or form submissions
<b>WebSettings</b>	Contains miscellaneous configuration options
<b>addJavascriptInterface(Object, String)</b>	Allows Java objects to be bound into the WebView so they can be controlled from the web page’s JavaScript.

*Table 16 — WebView customization points*

The following example illustrates a WebView managing event handling and interacting with JavaScript.

```
mWebView = new WebView(this);
setContentView(webView);

//make MyJavaScriptInterface accessible from javascript
webView.addJavaScriptInterface(new MyJavaScriptInterface(),
"android");
webView.loadUrl("http://developer.att.com/");

final class MyJavaScriptInterface {
    //invoked from javascript "android.someMethod()"
    public void someMethod() {
        mHandler.post(new Runnable() {
            //invoke a javascript method
            mWebView.loadUrl("javascript:doSomething()");
        })
    }
}
```

*Coding Example: WebView managing event handling and interacting with JavaScript.*

## 7. Best Practices

This section introduces some tips, tricks and best practices for developing Android applications, as well as things you'll need to know to about optimizing your application for the AT&T network.

### 7.1 User Interface

Creating a unified look and feel throughout your application's user interface makes your product look professional, and may make it more likely that a user will chose to download your application over another.

Here are a few tips to achieving a polished, professional look with your application:

#### DESIGN A PROFESSIONAL-LOOKING APPLICATION ICON

- Use the **Android Icon Templates Pack** as a starting point for your icon.
- Keep icon designs simple.
- Restrain the use of color in your icon. Note that base of a launcher icon should be grey and feel solid.
- Use the correct angles for the specific icon types, as described in the Android User Interface Guidelines.

#### MAKE PROPER USE OF THE BACK BUTTON

The BACK button will move backwards chronologically in the stack of activities. If your application is contained primarily within one screen, pressing the BACK button should bring the previously used application activity into the foreground, putting your application in the background.

If your application has multiple 'layers' or windows, such as the Maps application or the web browser, it may make more sense for the BACK button to restore a previous state within your application

#### CREATE EASY-TO-USE MENUS

Choosing the right type of menu for a given situation, and optimizing the number of items in a menu and placement of those items can make menus very intuitive and easy for the user.

- **Put only the most important commands fixed on the screen**  
The most important and most frequently used commands probably belong fixed to the user interface of your application, such as the 'Write New' button in an email application. Other commands used less frequently, or only available in some contexts and not others can go within the menus
- **Put global commands in the Options menu; selection-specific commands in the Context menu**  
Global commands for your application should go in the Options menu; commands related to the current selection should be in the Context menu (i.e. activated with a touch & hold)
- **List the most frequently used operations first**  
It's always a good idea to put the most frequently used commands first, since some device screens can be small and menus may scroll, which means all commands may not be initially visible and obvious.
- **Don't put commands *only* in a Context menu**  
Make sure users can 'discover' your application and figure out how to use it. A user should be able to fully use your application without resorting to context menus. Context menus are great time-savers, but not obvious to all users.
- **Use short, distinct names menus**  
Ensure your menu names are not truncated or ambiguous. If a text label in the Options icon menu is too long, the system truncates it in the middle. Thus, "Create Notification" is truncated to something like "Create...ication". Truncation should be avoided whenever possible, so it is best to keep commands short but descriptive.
- **Dim or hide menu items that are not available in the current context**  
Make sure commands not available in a given context are dimmed so that they do not appear as valid options to the user.

#### USE THE NOTIFICATION SYSTEM TO MESSAGE THE USER

Use the Android notification system if your background application needs to send an alert to the user, rather than a modal dialog which could interrupt an important user task. Standard Android notifications can be viewed at the user's convenience, and are much less likely to get in the way.

## 7.2 Optimizing Code for Android

Android applications should be optimized for efficiency. An efficient application strives for performance, but also balances this with the constraints of slower processors, smaller screens, constrained memory and attempts to preserve battery power as much as possible. When developing in the simulation environment on a workstation, it's easy to overlook these concerns, but getting them right and building an efficient application is crucial.

Code optimization should be considered a design task rather than a compilation/packaging task. That is, optimization and efficiency should be designed into the application from the ground up. Efficient code does more with fewer lines of instructions. It is clever, compact, and doesn't demand a large memory footprint.

***TIP:** There are two basic rules for resource-constrained systems:*

- Don't do work that you don't need to do*
- Don't allocate memory if you can avoid it*

### GENERAL OPTIMIZATION TIPS

The best way to write efficient code is to truly understand what the code really does, and evaluate whether there may be a simpler, faster and less 'expensive' way to achieve the same effect. Don't be concerned about using a memory or CPU intensive operation if you truly need it, but ensure these decisions are made with forethought.

Additional ideas include...

- Strive to write good programs rather than fast ones.
- Strive to avoid design decisions that limit performance.
- Consider performance consequences of API design decisions.
- Measure performance before and after each attempted optimization.
- Avoid creating objects.

- Use native methods.

### SAMPLE PERFORMANCE TIMES

The following table gives approximate run times for some common actions, to provide an illustration of the relative time cost of one operation over another. Consider the impact of each action when choosing (or not) to employ it.

Action	Time
Add a local variable	1
Add a member variable	4
Call String.length()	5
Call empty static native method	5
Call empty static method	12
Call empty virtual method	12.5
Call empty interface method	15
Call Iterator:next() on a HashMap	165
Call put() on a HashMap	600
Inflate 1 View from XML	22,000
Inflate 1 LinearLayout containing 1 TextView	25,000
Inflate 1 LinearLayout containing 6 View objects	100,000
Inflate 1 LinearLayout containing 6 TextView objects	135,000
Launch an empty activity	3,000,000

Table 17 — Sample performance times of some common actions

## 7.3 Localization

Android devices will run on many devices in regions all around the world, so it's important that applications can be easily adapted to other locales.

To make localization simple, Android applications use the resource framework, which keeps language resources separate from core Java functionality.

***TIP:** Most or all of the contents of an application's user interface can be put into resource files. The behavior of the user interface is driven by Java code, which does not belong in resource files. If, for example, user data needs to be formatted or sorted differently depending on locale, use Java to handle the data programmatically rather than storing this data in a locale resource file.*

## RESOURCE SWITCHING

Resources, such as text strings, layouts, sounds, graphics, and any other static data an Android application needs are loaded when an Android application is launched. Android automatically selects and loads the resources that best match the device.

Default text strings for an application must be stored in `res/values/strings.xml`. The default strings are loaded if a locale-specific version of strings cannot be found or is not defined. If the default strings cannot be found, the application will not run.

The default resource set must also include any default drawables and layouts, and can include other types of resources such as animations.

Default Resource	Description
<code>res/drawable/</code>	Required directory holding at least one graphic file, for the application's icon in the Android Market
<code>res/layout/</code>	Required directory holding an XML file that defines the default layout
<code>res/anim/</code>	Required if any <code>res/anim-&lt;qualifiers&gt;</code> folders exist
<code>res/xml/</code>	Required if any <code>res/xml-&lt;qualifiers&gt;</code> folders exist
<code>res/raw/</code>	Required if any any <code>res/raw-&lt;qualifiers&gt;</code> folders exist

Table 18 — Default resources to define for localization

***TIP:** Ensure each reference to an Android resource in the application code defines the default resource, and ensure that the default string file is complete: A localized string file can contain a subset of the strings, but the default string file must contain them all.*

## LOCALIZATION STRATEGIES

Every situation and operating environment cannot be anticipated when designing an application. This is particularly true where localization is concerned, since an application may run on devices in other regions which cannot be tested (practically, that is) by the developer beforehand. Therefore, when designing an application, ensure it will, at best, function normally; and, at worst, fail gracefully regardless of the device it runs on and the region it operates in.

Some additional tips include:

- Design applications to work in any locale.
- Ensure applications include a full set of default resources.
- Design a flexible layout.
- Avoid creating more resource files and text strings than you need.
- Test localized apps on the target device, or using the emulator.

## 7.4 Common Issues

A few tips might help you as you develop applications for Android.

### DEVICE VERSIONS

There is a wide variety of Android devices available, and planned in coming months. Devices vary in form-factor, hardware capabilities and screen resolutions. Additionally, the Operating System versions can vary across devices from version 1.5, version 1.6, 2.0 or later.

While the variety of devices gives great freedom of choice to the consumer, it means that application developers must consider how their applications will run on various devices and plan accordingly.

## JAVA STANDARDS

While Android supports many of the standard APIs in Java, it does not offer full compatibility with Java SE and Java ME. This means Java applications written for other platforms may need modification to run on Android.

## PERFORMANCE

Android handles Garbage Collection automatically. However, Garbage Collection can cause performance issues if it occurs when your application is making heavy use of the CPU. To avoid inadvertently kicking off the Garbage Collection process during performance-critical processes, be aware of the memory allocation requests your application is making, particularly when executing code which demands high performance, such as game rendering. Use the Allocation Tracker tool in the Android SDK to see stack traces which lead to memory allocation requests, and optimize these sections of code, if needed.

## 8. Android Development Tools and Android SDK

Most application developers will want to use an IDE (Integrated Development Environment) to write code for Android. Any IDE can be used (such as IntelliJ), or even a basic editor (such as Emacs), although Eclipse with the ADT (Android Development Tools) plug-in is the recommended method since it provides editing, building, debugging, packaging, and signing functionality all in one place.

If you are writing for Motorola devices in particular, you may wish to use the MOTODEV Studio as your IDE. MOTODEV Studio is a modified version of Eclipse + ADT plug-in with some additional plug-ins helpful for optimizing applications for unique hardware in some Motorola devices. Motorola's version of Eclipse can co-exist with any other versions of Eclipse on your development workstation, allowing other versions of Eclipse to be used for non-Motorola device application development.

### 8.1 Creating an Android Project

Using the Eclipse IDE with Android Development Tools (ADT) plug-in is the simplest way to set up your development environment, and offers the most integrated functionality. This is the development environment used internally at Google. Eclipse with ADT allows Android applications to be written and debugged faster and more easily. Here are a few of the benefits of using Eclipse with ADT:

- Eclipse + ADT gives the developer access to other Android development tools from inside the Eclipse IDE, for example to take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse.
- Eclipse + ADT provides a New Project Wizard to help quickly create and set up all of the basic files needed for a new Android application.
- Eclipse + ADT automates and simplifies the process of building Android applications.
- Eclipse + ADT provides an Android code editor that helps with writing valid XML for Android manifest and resource files.
- Eclipse + ADT will export projects into a signed APK, which can be distributed to users.

### 8.1.1 Creating an Android Project in Eclipse with ADT

The New Project Wizard dramatically simplifies project setup. To create a new project, simply do the following:

- Select **File > New > Project**,
- Select **Android > Android Project**, and click **Next**,
- Select the contents for the project:
  - Enter a *Project Name*. This will be the name of the folder where your project is created.
  - Under Contents, select **Create new project in workspace**. Select your project workspace location.
  - Under Target, select an Android target to be used as the project's Build Target. The Build Target specifies which Android platform you'd like your application built against.

***TIP:** Unless you know that you'll be using new APIs introduced in the latest SDK, you should select a target with the lowest platform version possible, such as Android 1.1.*

- Under Properties, fill in all necessary fields.
  - Enter an *Application name*. This is the human-readable title for your application — the name that will appear on the Android device.
  - Enter a *Package name*. This is the package namespace (following the same rules as for packages in the Java programming language) where all your source code will reside.
  - Select *Create Activity* and enter a name for your main Activity class.
  - Enter a *Min SDK Version*. This is an integer that indicates the minimum API Level required to properly run your application. Entering this here automatically sets the `minSdkVersion` attribute in the `<uses-sdk>` of your Android

Manifest file. If you're unsure of the appropriate API Level to use, copy the API Level listed for the Build Target you selected in the Target tab.

- Click **Finish**

Once you click Finish, the following files and directories will be created for your project automatically:

Component	Description
<b>src/</b>	Includes your stub Activity Java file. All other Java files for your application go in this directory
<b>&lt;Android Version&gt;/</b>	(e.g., Android 1.1/) Includes the android.jar file that your application will build against. This is determined by the build target that you have chosen in the New Project Wizard.
<b>gen/</b>	Contains the Java files generated by ADT, such as your <code>R.java</code> file and interfaces created from AIDL files
<b>assets/</b>	Initially empty. Used for storing raw asset files
<b>res/</b>	A folder for application resources, such as drawable files, layout files, string values, etc.
<b>AndroidManifest.xml</b>	The Android Manifest for your project
<b>default.properties</b>	This file contains project settings, such as the build target. This file is integral to the project, as such, it should be maintained in a Source Revision Control system. It should never be edited manually — to edit project properties, right-click the project folder and select "Properties"

Table 19 — Project files automatically generated by the ADT New Project Wizard

### 8.1.2 Creating an Android Project in Another IDE

When using IDEs other than Eclipse, some manual configuration is needed. You will need to be familiar with the following tools:

SDK Tool	Description
<b>android</b>	To create/update Android projects and to create/move/delete AVDs
<b>Android Emulator</b>	To run your Android applications on an emulated Android platform
<b>Android Debug</b>	To interface with your emulator or connected device (install

SDK Tool	Description
<b>Bridge</b>	apps, shell the device, issue commands, etc
<b>Ant</b>	To compile and build your Android project into an installable .apk file
<b>Keytool</b>	To generate a keystore and private key, used to sign your .apk file
<b>Jarsigner</b>	(Or similar) To sign your .apk file with a private key generated by keytool

Table 20 — SDK tools needed when using IDEs other than Eclipse + ADT

To create a project, run the android tool. It will create the following directories, default application files, stub files, configuration files and a build file:

Component	Description
<b>AndroidManifest.xml</b>	The application manifest file, synced to the specified Activity class for the project
<b>build.xml</b>	Build file for Ant
<b>default.properties</b>	Properties for the build system. Do not modify this file.
<b>build.properties</b>	Customizable properties for the build system. You can edit this file to override default build settings used by Ant and provide a pointer to your keystore and key alias so that the build tools can sign your application when built in release mode.
<b>src/your/package/namespace/ActivityName.java</b>	The Activity class you specified during project creation
<b>bin/</b>	Output directory for the build script
<b>gen/</b>	Holds Ant-generated files, such as R.java
<b>libs/</b>	Holds private libraries
<b>res/</b>	Holds project resources
<b>src/</b>	Holds source code
<b>tests/</b>	Holds a duplicate of all-of-the-above, for testing purposes

Table 21 — Items generated for a new project using the Android tool

Once a project has been created, you can begin development. You will use the Android Debug Bridge (adb) (located in the SDK tools/ directory) to send your application to the emulator.

## 9. Deployment

Once all the hard work of developing your application is complete, an exciting step is deploying your application so that the world can discover it. Several important steps must be completed in order to deploy your application, including digitally signing to prove authenticity.

### 9.1 Signing Your Application

All Android applications must be digitally signed before Android will install them on an emulator or an actual device. Applications are signed either with a debug key or with a private key. Debug keys are typically used for testing on an emulator. Private keys are used for deployment when the application is distributed to end-users.

If building in the Eclipse environment with the ADT plug-in, the plug-in will sign .apk files automatically with a debug key for the purposes of testing on an emulator or development device. This allows applications to be quickly and easily run and tested.

#### IMPORTANT POINTS TO REMEMBER ABOUT SIGNING

- All applications *must* be signed. Android will not install an application that is not signed
- You can use self-signed certificates to sign your applications. No certificate authority is needed
- When you are ready to release your application to end-users, you must sign it with a suitable private key. You cannot publish an application that is signed with the debug key generated by the SDK tools
- The system tests a signer certificate's expiration date only at install time. If an application's signer certificate expires after the application is installed, the application will continue to function normally.
- You can use standard tools — Keytool and Jarsigner — to generate keys and sign your application .apk files.
- Once you have signed the application, use the zipalign tool to optimize the final APK package.

### 9.1.1 Signing for Public Release

In preparation for releasing your application to the public, it must be signed. Here is an overview of the process:

#### 1. Obtain a Suitable Private Key

- The private key in your possession, which represents the personal, corporate, or organizational entity to be identified with the application,
- The private key has a validity period that exceeds the expected lifespan of the application or application suite. A validity period of more than 25 years is recommended (applications signed for Android Market must have a validity period ending after 22 October 2033),
- The private key is private — not the debug key generated by the Android SDK tools,
- The private key may be self-signed. If you do not have a suitable key, you must generate one using Keytool.

#### 2. Compile the Application In Release Mode

Compiling in Release Mode instructs the IDE not to sign the application with the debug key, which allows you to sign the application with your private key.

#### 3. Sign Your Application With Your Private Key

Sign the application with your private key, ensuring it has the attributes described in Step 1, above. Use the Jarsigner utility to do this.

#### 4. Align the Final APK Package

Once the application has been signed, use the zipalign tool to process the signed package. Zipalign ensures all uncompressed data starts with a particular byte alignment, which reduces the amount of RAM needed to load the application. Ensure the application is signed with a private key before running zipalign, otherwise the zipalign optimizations will be undone if the application is signed afterwards.

## 10. Porting Applications to Android

While Android is still relatively new as a platform, it is rapidly maturing and becoming widely adopted, making it an attractive target platform for porting existing applications.

### 10.1 J2ME Applications

Since Android runs applications in Java, existing Java ME applications can be run on, or ported to Android devices fairly simply.

There are two common approaches to running J2ME applications to Android:

- Use a J2ME emulator on Android to run the J2ME application, or
- Rewrite your application to use Android APIs natively.

#### EMULATORS

There are emulators available for Android in the Android Marketplace, for example MicroEmulator (<http://www.microemu.org/>) includes tools to convert JAD and JAR files into an apk which can be installed on an Android device or emulator. Opera mini was successfully ported to Android this way.

#### PORTING APPLICATIONS

Android runs Java, but does not support every API defined in the J2ME standard. However, many Java applications can be run on Android with little or no code changes.

It may seem daunting to rewrite your application, but there are distinct benefits:

- Applications written for Android run on all Android devices, plus the vast majority of Java-capable devices, meaning your application can virtually be run everywhere,
- Applications can have a more polished appearance by taking advantage of Android's User Interface APIs instead of using an add-on library such as LWUIT, and
- The learning process for J2ME developers is short and easy

## 10.2 Going Native

With the Android Native Development Kit (NDK), it's possible to embed, within your application, components written in other programming languages, such as C and C++. In some situations, it makes sense to run application components in their native languages, rather than porting them to Android's version of Java. For example, highly optimized, self-contained routines which do not require much RAM are good candidates, such as signal processing routines, physics simulations, and so on. These components might run better in their native form, or at least might not be worth the effort to port and optimize if the gains are small.

As an application developer, you must carefully evaluate when to use Android NDK, if at all. Applications which run C or C++ using the NDK are always more complex than pure Android applications. However, in some cases, running highly-optimized C or C++ code in the NDK can yield run-time performance benefits over Java code, in addition to the time savings associated with code reuse.

## 11. Conclusion

Android is a robust platform, giving developers the tools they need to write powerful, compelling and useful applications consumers will love.

We hope this paper has helped illustrate some of the possibilities for application development on the Android platform. For more information, and to get started, check out the Android documentation and SDK at <http://developer.android.com>.

## 12. Works Cited

**Android (Operating System)** [Online] / auth. Wikipedia // Wikipedia.org. - January 19, 2010. - January 19, 2010. - [http://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/Android_(operating_system)).

**Android and iPhone browser wars, Part 2: Build a browser-based application for iPhone and Android** [Online] / auth. Frank Ableson, IBM // <http://www.ibm.com/developerworks/>. - January 05, 2010. - February 11, 2010. - <http://www.ibm.com/developerworks/opensource/library/os-androidiphone2/index.html>.

**Android Developers** [Online] / auth. Google, Inc. // Android SDK and Reference. - 01 19, 2010. - 01 20, 2010. - <http://developer.android.com/index.html>.

**Android Porting Kit** [Online] // Viosoft Corporation. - 2009. - 02 22, 2010. - <http://99.9.168.19/solutions/android-porting-kit/>.

**Android WebView (WebKit)** [Online] / auth. Nazmul // developerlife – Tutorials » Android WebView (WebKit). - August 11, 2008. - 02 12, 2010. - <http://developerlife.com/tutorials/?p=369>.

**Flash Development with Android SDK 1.5** [Online] // FlashMobileBlog. - August 12, 2009. - 02 22, 2010. - <http://www.flashmobileblog.com>.

**iPhone SDK Comparison Chart** [Online] / auth. Patel Nilay // Engadget. - March 6, 2008. - January 20, 2010. - <http://www.engadget.com/2008/03/06/iphone-sdk-comparison-chart/>.

**SMS Messaging in Android** [Online] / auth. Lee Wei-Meng // mobiForge. - June 2009. - January 22, 2010. - <http://mobiforge.com/developing/story/sms-messaging-android>.

**WebKit - Wikipedia** [Online] / auth. Wikipedia // Wikipedia. - February 10, 2010. - 02 10, 2010. - <http://en.wikipedia.org/wiki/WebKit>.

**What is Android?** [Online] // Embetek.com. - July 6, 2009. - 02 19, 2010. - [http://www.embetek.com/index.php?option=com\\_content&view=article&id=49:android](http://www.embetek.com/index.php?option=com_content&view=article&id=49:android).

**Why Develop for Android?** [Online] / auth. M Michael // TrueSong Media. - November 12, 2009. - January 21, 2010. - <http://www.truesongmedia.com/2009/11/why-develop-for-android/>.