**Android**

**An Open Handset Alliance Project**

**Installing the SDK**

**Anatomy of an Android Application**

**Life Cycle of an Android Application**

**Package Index**

**By dbongman**

**Installing the SDK**

This page describes how to install the Android SDK and set up your development environment. If you haven't downloaded the SDK yet, you can use the link below to get started. Then read the rest of this document to learn how to install, configure, and use the SDK to create Android applications.

[**Download the SDK**](http://code.google.com/android/download.html)

**Upgrading?**

If you have already downloaded an earlier version of the SDK and developed applications on it, please read the [**Upgrading the SDK**](http://code.google.com/android/intro/upgrading.html) document instead. The *Upgrading* document explains how to migrate your existing development environment and applications to the new SDK release. The document also provides information that you will find useful during the upgrade, such as lists of framework and API changes.

**Contents**

[**System and Software Requirements**](http://code.google.com/intl/ko/android/intro/installing.html#developmentrequirements)

[**Installing the SDK**](http://code.google.com/intl/ko/android/intro/installing.html#installingsdk)

[**Installing the Eclipse Plugin (ADT)**](http://code.google.com/intl/ko/android/intro/installing.html#installingplugin)

[**Developing Android Applications on Eclipse**](http://code.google.com/intl/ko/android/intro/installing.html#developingwitheclipse)

[**Developing Android Applications with Other IDEs and Tools**](http://code.google.com/intl/ko/android/intro/installing.html#otherides)

[**Debugging**](http://code.google.com/intl/ko/android/intro/installing.html#debugging)

[**Debug and Test Settings on the Device**](http://code.google.com/intl/ko/android/intro/installing.html#additionaldebugging)

[**Top Debugging Tips**](http://code.google.com/intl/ko/android/intro/installing.html#toptips)

[**Building and Installing an Android Application**](http://code.google.com/intl/ko/android/intro/installing.html#building)

[**Removing an Android Application**](http://code.google.com/intl/ko/android/intro/installing.html#uninstalling)

[**Eclipse Tips**](http://code.google.com/intl/ko/android/intro/installing.html#tips)

**System and Software Requirements**

To develop Android applications using the code and tools in the Android SDK, you need a suitable development computer and development environment, as described below.

*Supported Operating Systems*

* Windows XP or Vista
* Mac OS X 10.4.8 or later (x86 only)
* Linux (tested on Linux Ubuntu Dapper Drake)

*Supported Development Environments*

* Eclipse IDE
	+ [Eclipse](http://www.eclipse.org/downloads/) 3.2, 3.3 (Europa)
		- Eclipse [JDT](http://www.eclipse.org/jdt) plugin (included in most Eclipse IDE packages)
		- [WST](http://www.eclipse.org/webtools) (optional, but needed for the Android Editors feature; included in [most Eclipse IDE packages](http://www.eclipse.org/downloads/moreinfo/compare.php))
	+ [JDK 5 or JDK 6](http://java.sun.com/javase/downloads/index.jsp) (JRE alone is not sufficient)
	+ Not compatible with Gnu Compiler for Java (gcj)
	+ [Android Development Tools plugin](http://code.google.com/intl/ko/android/intro/installing.html#installingplugin) (optional)
* Other development environments or IDEs
	+ [JDK 5 or JDK 6](http://java.sun.com/javase/downloads/index.jsp) (JRE alone is not sufficient)
	+ Not compatible with Gnu Compiler for Java (gcj)
	+ [Apache Ant](http://ant.apache.org/) 1.6.5 or later for Linux and Mac, 1.7 or later for Windows

Note: If JDK is already installed on your development computer, please take a moment to make sure that it meets the version requirements listed above. In particular, note that some Linux distributions may include JDK 1.4 or Gnu Compiler for Java, both of which are not supported for Android development.

**Installing the SDK**

After downloading the SDK, unpack the .zip archive to a suitable location on your machine. By default, the SDK files are unpacked into a directory named android\_sdk\_*<platform*>\_*<release>*\_*<build>*. The directory contains the subdirectories tools/, samples/, and others.

Make a note of the name and location of the unpacked SDK directory on your system — you will need to refer to the SDK directory later, when setting up the Android plugin or using SDK tools.

Optionally, you can add the path to the SDK tools directory to your path. As mentioned above, the tools/ directory is located in the SDK directory.

* On Linux, edit your ~/.bash\_profile or ~/.bashrc file. Look for a line that sets the PATH environment variable and add the full path to the tools/ directory to it. If you don't see a line setting the path, you can add one:

export PATH=${PATH}:*<your\_sdk\_dir>*/tools

* On a Mac, look in your home directory for .bash\_profile and proceed as for Linux. You can create the .bash\_profile, if you haven't already set one up on your machine.
* On Windows, right click on My Computer, and select Properties. Under the Advanced tab, hit the Environment Variables button, and in the dialog that comes up, double-click on Path under System Variables. Add the full path to the tools/ directory to the path.

Adding tools to your path lets you run Android Debug Bridge (adb) and the other command line [tools](http://code.google.com/android/intro/tools.html) without needing to supply the full path to the tools directory. Note that, if you update your SDK, you should remember to update your PATH settings to point to the new location, if different.

**Installing the Eclipse Plugin (ADT)**

**Tips for downloading ADT**

If you are having trouble downloading the ADT plugin after following the steps listed at left, here are some suggestions:

* In Step 4, try changing the remote update site URL to use http, rather than https.
* If you are behind a firewall (such as a corporate firewall), make sure that you have properly configured your proxy settings in Eclipse. In Eclipse 3.3, you can configure proxy information from the main Eclipse menu in **Window** > **Preferences** > **General** > **Network Connections**. In Eclipse 3.2, use **Window** > **Preferences** > **Install/Update**.

If you are still unable to use Eclipse to download the ADT plugin, follow these steps to download and install the plugin from your computer:

1. [Download the ADT zip file](http://code.google.com/android/adt_download.html)
2. Unpack the zip file in its own folder
3. Follow steps 1 and 2 in the default install instructions (on the left)
4. Press **New Local Site...**
5. In the resulting dialog box, select the folder where you unpacked the zip file
6. Follow steps 5 through 11 in the default install instructions (on the left) to complete the installation

Note that to update your plugin, you will have to follow these steps again instead of the default update instructions.

If you will be using the Eclipse IDE as your environment for developing Android applications, you can install a custom plugin called Android Development Tools (ADT), which adds integrated support for Android projects and tools. The ADT plugin includes a variety of powerful extensions that make creating, running, and debugging Android applications faster and easier.

If you *will not* be using the Eclipse IDE, you do not need to download or install the ADT plugin.

To download and install the ADT plugin, follow the steps below.

1. Start Eclipse, then select **Help** > **Software Updates** > **Find and Install...**.
2. In the dialog that appears, select **Search for new features to install** and press **Next**.
3. Press **New Remote Site**.
4. In the resulting dialog box, enter a name for the remote site (e.g. Android Plugin) and enter this as its URL:

https://dl-ssl.google.com/android/eclipse/

Press **OK**.

1. You should now see the new site added to the search list (and checked). Press **Finish**.
2. In the subsequent Search Results dialog box, select the checkbox for **Android Plugin** > **Developer Tools**. This will check both features: "Android Developer Tools", and "Android Editors". The Android Editors feature is optional, but recommended. If you choose to install it, you need the WST plugin mentioned earlier in this page.

Now press **Next**.

1. Read the license agreement and then select **Accept terms of the license agreement**, if appropriate. Press **Next**.
2. Press **Finish**.
3. The ADT plugin is not signed; you can accept the installation anyway by pressing **Install All**.
4. Restart Eclipse.
5. After restart, **update your Eclipse preferences** to point to the SDK directory:
	1. Select **Window** > **Preferences...** to open the Preferences panel. (Mac OS X: **Eclipse** > **Preferences**)
	2. Select **Android** from the left panel.
	3. For the SDK Location in the main panel, press **Browse...** and locate the SDK directory.
	4. Press **Apply**, then **OK**.

Note that the "Android Editors" feature of ADT requires several optional Eclipse components (for example, WST). If you encounter an error when installing ADT, your Eclipse installion might not include those components. For information about how to quickly add the necessary components to your Eclipse installation, see the troubleshooting topic [ADT Installation Error: "requires plug-in org.eclipse.wst.sse.ui"](http://code.google.com/android/kb/troubleshooting.html#installeclipsecomponents).

**Updating the ADT Plugin**

In some cases, a new ADT plugin may become available for your existing version of the SDK. You can use the steps below to update the ADT plugin from inside Eclipse.

1. Select **Help** > **Software Updates** > **Find and Install...**.
2. Select **Search for updates of the currently installed features** and press **Finish**.
3. If any update for ADT is available, select and install.

Alternatively,

1. Select **Help** > **Software Updates** > **Manage Configuration**.
2. Navigate down the tree and select **Android Development Tools <version>**
3. Select **Scan for Updates** under **Available Tasks**.

**Developing Android Applications on Eclipse**

To begin developing Android applications in the Eclipse IDE, you first create an Android project and then set up a launch configuration. After that, you can write, run, and debug your application.

The sections below provide instructions assuming that you have installed the ADT plugin in your Eclipse environment. If you haven't installed the ADT plugin, you should do that before using the instructions below. See the [Installing the Eclipse Plugin (ADT)](http://code.google.com/intl/ko/android/intro/installing.html#installingplugin) for more information.

**Creating an Android Project**

The ADT plugin provides a New Project Wizard that you can use to quickly create an Eclipse project for new or existing code. To create the project, follow these steps:

1. Select **File** > **New** > **Project**
2. Select **Android** > **Android Project**, and press **Next**
3. Select the contents for the project:
	* Select **Create new project in workspace** to start a project for new code.

Enter the project name, the base package name, the name of a single Activity class to create as a stub .java file, and a name to use for your application.

* + Select **Create project from existing source** to start a project from existing code. Use this option if you want to build and run any of the sample applications included with the SDK. The sample applications are located in the samples/ directory in the SDK.

Browse to the directory containing the existing source code and click OK. If the directory contains a valid Android manifest file, the ADT plugin fills in the package, activity, and application names for you.

1. Press **Finish**.

The ADT plugin creates the these folders and files for you as appropriate for the type of project:

* src/   A folder that includes your stub .java Activity file.
* res/   A folder for your resources.
* AndroidManifest.xml   The manifest for your project.

**Creating a Launch Configuration**

Before you can run and debug your application in Eclipse, you must create a launch configuration for it. A launch configuration specifies the project to launch, the Activity to start, the emulator options to use, and so on.

To create a launch configuration for the application, follow these steps:

1. Select **Run** > **Open Run Dialog...** or **Run** > **Open Debug Dialog...** as appropriate.
2. In the project type list on the left, right-click **Android Application** and select **New**.
3. Enter a name for your configuration.
4. On the Android tab, browse for the project and Activity to start.
5. On the Target tab, set the desired screen and network properties, as well as any other [emulator startup options](http://code.google.com/android/reference/emulator.html#startup-options).
6. You can set additional options on the Common tab as desired.
7. Press **Apply** to save the launch configuration, or press **Run** or **Debug** (as appropriate).

**Running and Debugging an Application**

Once you've set up the project and launch configuration for your application, you can run or debug it as described below.

From the Eclipse main menu, select **Run** > **Run** or **Run** > **Debug** as appropriate, to run or debug the active launch configuration.

Note that the active launch configuration is the one most recently selected in the Run configuration manager. It does not necessarily correspond to the application that is selected in the Eclipse Navigation pane (if any).

To set or change the active launch configuration, use the Run configuration manager, which you can access through **Run** > **Open Run Dialog...** or **Run** > **Open Debug Dialog...**.

Running or debugging the application triggers these actions:

* Starts the emulator, if it is not already running.
* Compiles the project, if there have been changes since the last build, and installs the application on the emulator.
* **Run** starts the application.
* **Debug** starts the application in "Wait for debugger" mode, then opens the Debug perspective and attaches the Eclipse Java debugger to the application.

**Developing Android Applications with Other IDEs and Tools**

The recommended way to develop an Android application is to use [Eclipse with the Android plugin](http://code.google.com/intl/ko/android/intro/installing.html#developingwitheclipse). This plugin provides editing, building, and debugging functionality integrated right into the IDE. However, the SDK includes tools to enable you to develop with other IDEs, including intelliJ (or if you'd rather use Eclipse without the plugin).

**Creating an Android Project**

The Android SDK includes activityCreator, a program that generates a number of stub files for your project, as well as a build file. You can use the program to create an Android project for new code or from existing code, such as the sample applications included in the SDK. For Linux and Mac, the SDK provides activityCreator.py, a Python script, and for Windows, activityCreator.bat, a batch script. Regardless of platform, you can use activityCreator in the same way.

To run activityCreator and create an Android project, follow these steps:

1. In the command line, change to the tools/ directory of the SDK and create a new directory for your project files. If you are creating a project from existing code, change to the root folder of your application instead.
2. Run activityCreator. In the command, you must specify a fully-qualified class name as an argument. If you are creating a project for new code, the class represents the name of a stub class that the script will create. If you are creating a project from existing code, you must specify the name of one Activity class in the package. Command options for the script include:
	* --out <folder> which sets the output directory. By default, the output directory is the current directory. If you created a new directory for your project files, use this option to point to it.
	* --ide intellij, which generates IntelliJ IDEA project files in the newly created project

Here's an example:

~/android\_linux\_sdk/tools $ ./activityCreator.py --out myproject your.package.name.ActivityName

package: your.package.name

out\_dir: myproject

activity\_name: ActivityName

~/android\_linux\_sdk/tools $

The activityCreator script generates the following files and directories (but will not overwrite existing ones):

* AndroidManifest.xml The application manifest file, synced to the specified Activity class for the project.
* build.xml An Ant file that you can use to build/package the application.
* src*/your/package/name/ActivityName*.java The Activity class you specified on input.
* *your\_activity*.iml, *your\_activity*.ipr, *your\_activity*.iws    [*only with the -ide intelliJ flag*] intelliJ project files.
* res/   A directory to hold resources.
* src/    The source directory.
* bin/    The output directory for the build script.

You can now move your folder wherever you want for development, but keep in mind that you'll have to use the [adb](http://code.google.com/android/reference/adb.html) program in the tools/ folder to send files to the emulator, so you'll need access between your solution and the tools/ folder.

Also, you should refrain from moving the location of the SDK directory, since this will break the build scripts (they will need to be manually updated to reflect the new SDK location before they will work again).

**Building an Android Application**

Use the Ant build.xml file generated by activityCreator to build your application.

1. If you don't have it, you can obtain Ant from the [Apache Ant home page](http://ant.apache.org/). Install it and make sure it is on your executable path.
2. Before calling Ant, you need to declare the JAVA\_HOME environment variable to specify the path to where the JDK is installed.

Note: When installing JDK on Windows, the default is to install in the "Program Files" directory. This location will cause ant to fail, because of the space. To fix the problem, you can specify the JAVA\_HOME variable like this: set JAVA\_HOME=c:\Prora~1\Java\. The easiest solution, however, is to install JDK in a non-space directory, for example: c:\java\jdk1.6.0\_02.

1. If you have not done so already, follow the instructions for Creating a New Project above to set up the project.
2. You can now run the Ant build file by simply typing ant in the same folder as the build.xml file for your project. Each time you change a source file or resource, you should run ant again and it will package up the latest version of the application for you to deploy.

**Running an Android Application**

To run a compiled application, you will upload the .apk file to the /data/app/ directory in the emulator using the [adb](http://code.google.com/android/reference/adb.html) tool as described here:

1. Start the emulator (run *<your\_sdk\_dir>*/tools/emulator from the command line)
2. On the emulator, navigate to the home screen (it is best not to have that application running when you reinstall it on the emulator; press the **Home** key to navigate away from that application).
3. Run adb install *myproject*/bin/<*appname*>.apk to upload the executable. So, for example, to install the Lunar Lander sample, navigate in the command line to *<your\_sdk\_dir>*/sample/LunarLander and type ../../tools/adb install bin/LunarLander.apk
4. In the emulator, open the list of available applications, and scroll down to select and start your application.

**Note:** When you install an Activity for the first time, you might have to restart the emulator before it shows up in the application launcher, or other applications can call it. This is because the package manager usually only examines manifests completely on emulator startup.

**Attaching a Debugger to Your Application**

This section describes how to display debug information on the screen (such as CPU usage), as well as how to hook up your IDE to debug running applications on the emulator.

Attaching a debugger is automated using the Eclipse plugin, but you can configure other IDEs to listen on a debugging port to receive debugging information.

1. **Start the** [**Dalvik Debug Monitor Server (DDMS) tool**](http://code.google.com/android/reference/ddms.html) **,** which acts as a port forwarding service between your IDE and the emulator.
2. **Set optional debugging configurations on your emulator**, such as blocking application startup for an activity until a debugger is attached. Note that many of these debugging options can be used without DDMS, such as displaying CPU usage or screen refresh rate on the emulator.
3. **Configure your IDE to attach to port 8700 for debugging.** We include information on [how to set up Eclipse to debug your project](http://code.google.com/intl/ko/android/intro/installing.html#eclipse).

**Configuring your IDE to attach to the debugging port**

DDMS will assign a specific debugging port to every virtual machine that it finds on the emulator. You must either attach your IDE to that port (listed on the Info tab for that VM), or you can use a default port 8700 to connect to whatever application is currently selected on the list of discovered virtual machines.

Your IDE should attach to your application running on the emulator, showing you its threads and allowing you to suspend them, inspect their state, and set breakpoints. If you selected "Wait for debugger" in the Development settings panel the application will run when Eclipse connects, so you will need to set any breakpoints you want before connecting.

Changing either the application being debugged or the "Wait for debugger" option causes the system to kill the selected application if it is currently running. You can use this to kill your application if it is in a bad state by simply going to the settings and toggling the checkbox.

**Debugging**

Android has a fairly extensive set of tools to help you debug your programs:

* [**DDMS**](http://code.google.com/android/reference/ddms.html) - A graphical program that supports port forwarding (so you can set up breakpoints in your code in your IDE), screen captures on the emulator, thread and stack information, and many other features. You can also run logcat to retrieve your Log messages. See the linked topic for more information.
* [**logcat**](http://code.google.com/android/reference/ddms.html#logcat) - Dumps a log of system messages. The messages include a stack trace when the emulator throws an error, as well as Log messages. To run logcat, see the linked topic.
* ...
* I/MemoryDealer( 763): MemoryDealer (this=0x54bda0): Creating 2621440 bytes heap at 0x438db000
* I/Logger( 1858): getView() requesting item number 0
* I/Logger( 1858): getView() requesting item number 1
* I/Logger( 1858): getView() requesting item number 2
* D/ActivityManager( 763): Stopping: HistoryRecord{409dbb20 com.google.android.home.AllApps}

...

* [**Android Log**](http://code.google.com/android/reference/android/util/Log.html)- A logging class to print out messages to a log file on the emulator. You can read messages in real time if you run logcat on DDMS (covered next). Add a few logging method calls to your code.

To use the Log class, you just call Log.v() (verbose), Log.d() (debug), Log.i() (information), Log.w() (warning) or Log.e (error) depending on the importance you wish to assign the log message.

Log.i("MyActivity", "MyClass.getView() — Requesting item number " + position)

You can use logcat to read these messages

* [**Traceview**](http://code.google.com/android/reference/traceview.html)- Android can save a log of method calls and times to a logging file that you can view in a graphical reader called Traceview. See the linked topic for more information.
* [**Eclipse plugin**](http://code.google.com/intl/ko/android/intro/installing.html#developingwitheclipse) - The Eclipse Android plugin incorporates a number of these tools (ADB, DDMS, logcat output, and other functionality). See the linked topic for more information.
* **Debug and Test Device Settings** - Android exposes several settings that expose useful information such as CPU usage and frame rate. See [Debug and Test Settings on the Emulator](http://code.google.com/intl/ko/android/intro/installing.html#additionaldebugging) below.

Also, see the [Troubleshooting](http://code.google.com/android/kb/troubleshooting.html) section of the doc to figure out why your application isn't appearing on the emulator, or why it's not starting.

**Debug and Test Settings on the Device**

Android lets you set a number of settings that will make it easier to test and debug your applications. To get to the development settings page on the emulator, go to **Dev Tools** > **Development Settings**. This will open the development settings page with the following options (among others):

* **Debug app**   Selects the application that will be debugged. You do not need to set this to attach a debugger, but setting this value has two effects:
	+ It will prevent Android from throwing an error if you pause on a breakpoint for a long time while debugging.
	+ It will enable you to select the *Wait for Debugger* option to pause application startup until your debugger attaches (described next).
* **Wait for debugger**    Blocks the selected application from loading until a debugger attaches. This way you can set a breakpoint in onCreate(), which is important to debug the startup process of an Activity. When you change this option, any currently running instances of the selected application will be killed. In order to check this box, you must have selected a debug application as described in the previous option. You can do the same thing by adding [waitForDebugger()](http://code.google.com/android/reference/android/os/Debug.html#waitForDebugger()) to your code.
* **Immediately destroy activities**   Tells the system to destroy an activity as soon as it is stopped (as if Android had to reclaim memory).  This is very useful for testing the [onFreeze(Bundle)](http://code.google.com/android/reference/android/app/Activity.html#onFreeze(android.os.Bundle)) / [onCreate(android.os.Bundle)](http://code.google.com/android/reference/android/app/Activity.html#onCreate(android.os.Bundle)) code path, which would otherwise be difficult to force. Choosing this option will probably reveal a number of problems in your application due to not saving state.
* **Show screen updates**    Flashes a momentary pink rectangle on any screen sections that are being redrawn. This is very useful for discovering unnecessary screen drawing.
* **Show CPU usage**   Displays CPU meters at the top of the screen, showing how much the CPU is being used. The top red bar shows overall CPU usage, and the green bar underneath it shows the CPU time spent in compositing the screen. *Note: You cannot turn this feature off once it is on, without restarting the emulator.*
* **Show background**   Displays a background pattern when no activity screens are visible. This typically does not happen, but can happen during debugging.

These settings will be remembered across emulator restarts.

**Top Debugging Tips**

**Quick stack dump**

To obtain a stack dump from emulator, you can log in with adb shell, use "ps" to find the process you want, and then "kill -3 ". The stack trace appears in the log file.

**Displaying useful info on the emulator screen**

The device can display useful information such as CPU usage or highlights around redrawn areas. Turn these features on and off in the developer settings window as described in [Setting debug and test configurations on the emulator](http://code.google.com/intl/ko/android/intro/installing.html#additionaldebugging).

**Getting system state information from the emulator (dumpstate)**

You can access dumpstate information from the Dalvik Debug Monitor Service tool. See [dumpsys and dumpstate](http://code.google.com/android/reference/adb.html#dumpsys) on the adb topic page.

**Getting application state information from the emulator (dumpsys)**

You can access dumpsys information from the Dalvik Debug Monitor Service tool. See [dumpsys and dumpstate](http://code.google.com/android/reference/adb.html#dumpsys) on the adb topic page.

**Getting wireless connectivity information**

You can get information about wireless connectivity using the Dalvik Debug Monitor Service tool. From the **Device** menu, select "Dump radio state".

**Logging Trace Data**

You can log method calls and other tracing data in an activity by calling android.os.Debug.startMethodTracing(). See [Running the Traceview Debugging Program](http://code.google.com/android/reference/traceview.html) for details.

**Logging Radio Data**

By default, radio information is not logged to the system (it is a lot of data). However, you can enable radio logging using the following commands:

adb shell

logcat -b radio

**Running adb**

Android ships with a tool called adb that provides various capabilities, including moving and syncing files to the emulator, forwarding ports, and running a UNIX shell on the emulator. See [Using adb](http://code.google.com/android/reference/adb.html) for details.

**Getting screen captures from the emulator**

Dalvik Debug Monitor Server (DDMS) can capture screenshots from the emulator.

**Using debugging helper classes**

Android provides debug helper classes such as [util.Log](http://code.google.com/android/reference/android/util/Log.html) and [Debug](http://code.google.com/android/reference/android/os/Debug.html) for your convenience.

**Building and Installing an Android Application**

Android requires custom build tools to be able to properly build the resource files and other parts of an Android application. Because of this, you must have a specialized build environment for your application.

Custom Android compilation steps include compiling the XML and other resource files, and creating the proper output format. A compiled Android application is an .apk file, which is a compressed file containing [.dex](http://code.google.com/android/reference/glossary.html) files, resource files, raw data files, and other files. You can create a properly structured Android project either from scratch, or from existing source files.

Android does not currently support development of third party applications in native code (C/C++).

**The recommended way** to develop an Android application is to [use Eclipse with the Android plugin](http://code.google.com/intl/ko/android/intro/installing.html#developingwitheclipse), which provides support for building, running, and debugging Android applications.

**If you have another IDE**, [Android provides tools for other IDEs](http://code.google.com/intl/ko/android/intro/installing.html#otherides) to build and debug Android applications, but they are not as integrated.

**Removing an Android Application**

To remove an application that you have installed on the emulator, you will need to [run adb](http://code.google.com/android/reference/adb.html) and delete the .apk file you sent to the emulator when you installed it. Use adb shell to drop into a shell on the device as described in the linked topic, navigate to data/app/, and then remove the file using rm *your\_app*.apk.

**Eclipse Tips**

**Executing arbitrary Java expressions in Eclipse**

You can execute arbitrary code when paused at a breakpoint in Eclipse. For example, when in a function with a String argument called "zip", you can get information about packages and call class methods. You can also invoke arbitrary static methods: for example, entering android.os.Debug.startMethodTracing() will start dmTrace.

Open a code execution window, select **Window**>**Show View**>**Display** from the main menu to open the Display window, a simple text editor. Type your expression, highlight the text, and click the 'J' icon (or CTRL + SHIFT + D) to run your code. The code runs in the context of the selected thread, which must be stopped at a breakpoint or single-step point. (If you suspend the thread manually, you have to single-step once; this doesn't work if the thread is in Object.wait().)

If you are currently paused on a breakpoint, you can simply highlight and execute a piece of source code by pressing CTRL + SHIFT + D.

You can highlight a block of text within the same scope by pressing ALT +SHIFT + UP ARROW to select larger and larger enclosing blocks, or DOWN ARROW to select smaller blocks.

Here are a few sample inputs and responses in Eclipse using the Display window.

|  |  |
| --- | --- |
| **Input** | **Response** |
| zip | (java.lang.String) /work/device/out/linux-x86-debug/android/app/android\_sdk.zip |
| zip.endsWith(".zip") | (boolean) true |
| zip.endsWith(".jar") | (boolean) false |

You can also execute arbitrary code when not debugging by using a scrapbook page. Search the Eclipse documentation for "scrapbook".

**Running DDMS Manually**

Although the recommended way to debug is to use the ADT plugin, you can manually run DDMS and configure Eclipse to debug on port 8700. (**Note:** Be sure that you have first started [DDMS](http://code.google.com/android/reference/ddms.html)).

**Anatomy of an Android Application**

There are four building blocks to an Android application:

* Activity
* Intent Receiver
* Service
* Content Provider

Not every application needs to have all four, but your application will be written with some combination of these.

Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are. See the [Android manifest file documentation](http://code.google.com/android/devel/bblocks-manifest.html) for complete details.

**Activity**

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the [Activity](http://code.google.com/android/reference/android/app/Activity.html) base class. Your class will display a user interface composed of [Views](http://code.google.com/android/reference/android/view/View.html) and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity. Moving to another screen is accomplished by a starting a new activity. In some cases an activity may return a value to the previous activity -- for example an activity that lets the user pick a photo would return the chosen photo to the caller.

When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

**Intent and Intent Filters**

Android uses a special class called an [Intent](http://code.google.com/android/reference/android/content/Intent.html) to move from screen to screen. An intent describes what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a URI. For example, to view contact information for a person, you would create an intent with the VIEW action and the data set to a URI representing that person.

There is a related class called an [IntentFilter](http://code.google.com/android/reference/android/content/IntentFilter.html). While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. An activity that is able to display contact information for a person would publish an IntentFilter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their IntentFilters in the AndroidManifest.xml file.

Navigating from screen to screen is accomplished by resolving intents. To navigate forward, an activity calls [startActivity(myIntent)](http://code.google.com/android/reference/android/app/Activity.html#startActivity(android.content.Intent)). The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches myIntent. The new activity is informed of the intent, which causes it to be launched. The process of resolving intents happens at run time when startActivity is called, which offers two key benefits:

* Activities can reuse functionality from other components simply by making a request in the form of an Intent
* Activities can be replaced at any time by a new Activity with an equivalent IntentFilter

**Intent Receiver**

You can use an [IntentReceiver](http://code.google.com/android/reference/android/content/IntentReceiver.html) when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may use the [NotificationManager](http://code.google.com/android/reference/android/app/NotificationManager.html) to alert the user if something interesting has happened. Intent receivers are registered in AndroidManifest.xml, but you can also register them from code using [Context.registerReceiver()](http://code.google.com/android/reference/android/content/Context.html#registerReceiver(android.content.IntentReceiver, android.content.IntentFilter)). Your application does not have to be running for its intent receivers to be called; the system will start your application, if necessary, when an intent receiver is triggered. Applications can also send their own intent broadcasts to others with [Context.broadcastIntent()](http://code.google.com/android/reference/android/content/Context.html#broadcastIntent(android.content.Intent)).

**Service**

A [Service](http://code.google.com/android/reference/android/app/Service.html) is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using [Context.startService()](http://code.google.com/android/reference/android/content/Context.html#startService(android.content.Intent, android.os.Bundle)) to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading [Life Cycle of an Android Application](http://code.google.com/android/intro/lifecycle.html).) Note that you can connect to a service (and start it if it's not already running) with the [Context.bindService()](http://code.google.com/android/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

**Content Provider**

Applications can store their data in files, an SQLite database, or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

To get more details on content providers, see [Accessing Content Providers](http://code.google.com/android/devel/data/contentproviders.html).

**Life Cycle of an Android Application**

In most cases, every Android application runs in its own Linux process. This process is created for the application when some of its code needs to be run, and will remain running until it is no longer needed *and* the system needs to reclaim its memory for use by other applications.

An unusual and fundamental feature of Android is that an application process's lifetime is *not* directly controlled by the application itself. Instead, it is determined by the system through a combination of the parts of the application that the system knows are running, how important these things are to the user, and how much overall memory is available in the system.

It is important that application developers understand how different application components (in particular [Activity](http://code.google.com/android/reference/android/app/Activity.html), [Service](http://code.google.com/android/reference/android/app/Service.html), and [IntentReceiver](http://code.google.com/android/reference/android/content/IntentReceiver.html)) impact the lifetime of the application's process. **Not using these components correctly can result in the system killing the application's process while it is doing important work.**

A common example of a process life-cycle bug is an [IntentReceiver](http://code.google.com/android/reference/android/content/IntentReceiver.html) that starts a thread when it receives an Intent in its [onReceiveIntent()](http://code.google.com/android/reference/android/content/IntentReceiver.html#onReceiveIntent(android.content.Context, android.content.Intent)) method, and then returns from the function. Once it returns, the system considers that IntentReceiver to be no longer active, and thus its hosting process no longer needed (unless other application components are active in it). Thus, it may kill the process at any time to reclaim memory, terminating the spawned thread that is running in it. The solution to this problem is to start a [Service](http://code.google.com/android/reference/android/app/Service.html) from the IntentReceiver, so the system knows that there is still active work being done in the process.

To determine which processes should be killed when low on memory, Android places them into an "importance hierarchy" based on the components running in them and the state of those components. These are, in order of importance:

1. A **foreground process** is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:
	* It is running an [Activity](http://code.google.com/android/reference/android/app/Activity.html) at the top of the screen that the user is interacting with (its [onResume()](http://code.google.com/android/reference/android/app/Activity.html#onResume()) method has been called).
	* It has an [IntentReceiver](http://code.google.com/android/reference/android/content/IntentReceiver.html) that is currently running (its [IntentReceiver.nReceiveIntent()](http://code.google.com/android/reference/android/content/IntentReceiver.html#onReceiveIntent(android.content.Context, android.content.Intent)) method is executing).
	* It has a [Service](http://code.google.com/android/reference/android/app/Service.html) that is currently executing code in one of its callbacks ([Service.onCreate()](http://code.google.com/android/reference/android/app/Service.html#onCreate()), [Service.onStart()](http://code.google.com/android/reference/android/app/Service.html#onStart(int, android.os.Bundle)), or [Service.onDestroy()](http://code.google.com/android/reference/android/app/Service.html#onDestroy())).

There will only ever be a few such processes in the system, and these will only be killed as a last resort if memory is so low that not even these processes can continue to run. Generally at this point the device has reached a memory paging state, so this action is required in order to keep the user interface responsive.

1. A **visible process** is one holding an [Activity](http://code.google.com/android/reference/android/app/Activity.html) that is visible to the user on-screen but not in the foreground (its [onPause()](http://code.google.com/android/reference/android/app/Activity.html#onPause()) method has been called). This may occur, for example, if the foreground activity has been displayed with a dialog appearance that allows the previous activity to be seen behind it. Such a process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running.
2. A **service process** is one holding a [Service](http://code.google.com/android/reference/android/app/Service.html) that has been started with the [startService()](http://code.google.com/android/reference/android/content/Context.html#startService(android.content.Intent, android.os.Bundle)) method. Though these processes are not directly visible to the user, they are generally doing things that the user cares about (such as background mp3 playback or background network data upload or download), so the system will always keep such processes running unless there is not enough memory to retain all foreground and visible process.
3. A **background process** is one holding an [Activity](http://code.google.com/android/reference/android/app/Activity.html) that is not currently visible to the user (its [onStop()](http://code.google.com/android/reference/android/app/Activity.html#onStop()) method has been called). These processes have no direct impact on the user experience. Provided they implement their activity life cycle correctly (see [Activity](http://code.google.com/android/reference/android/app/Activity.html) for more details), the system can kill such processes at any time to reclaim memory for one of the three previous processes types. Usually there are many of these processes running, so they are kept in an LRU list to ensure the process that was most recently seen by the user is the last to be killed when running low on memory.
4. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component of its application needs to run. As such, the system will often kill these processes in order to balance overall system resources between these empty cached processes and the underlying kernel caches.

When deciding how to classify a process, the system picks the most important level of all the components currently active in the process. See the [Activity](http://code.google.com/android/reference/android/app/Activity.html), [Service](http://code.google.com/android/reference/android/app/Service.html), and [IntentReceiver](http://code.google.com/android/reference/android/content/IntentReceiver.html) documentation for more detail on how each of these components contribute to the overall life cycle of a process. The documentation for each of these classes describes in more detail how they impact the overall life cycle of their application.

A process's priority may also be increased based on other dependencies a process has to it. For example, if process A has bound to a [Service](http://code.google.com/android/reference/android/app/Service.html) with the [Context.BIND\_AUTO\_CREATE](http://code.google.com/android/reference/android/content/Context.html#BIND_AUTO_CREATE) flag or is using a [ContentProvider](http://code.google.com/android/reference/android/content/ContentProvider.html) in process B, then process B's classification will always be at least as important as process A's.

**Package Index**

These are the Android APIs.

|  |  |
| --- | --- |
| [android](http://code.google.com/android/reference/android/package-summary.html)  | Contains the resource classes used by standard Android applications.  |
| [android.app](http://code.google.com/android/reference/android/app/package-summary.html)  | High-level classes encapsulating the overall Android application model.  |
| [android.awt](http://code.google.com/android/reference/android/awt/package-summary.html)  | Provides non-widget parts of the Abstract Windowing Tolkit (AWT).  |
| [android.content](http://code.google.com/android/reference/android/content/package-summary.html)  | Contains classes for accessing and publishing data on the device.  |
| [android.content.pm](http://code.google.com/android/reference/android/content/pm/package-summary.html)  |   |
| [android.database](http://code.google.com/android/reference/android/database/package-summary.html)  | Contains classes to explore data returned through a content provider.  |
| [android.database.sqlite](http://code.google.com/android/reference/android/database/sqlite/package-summary.html)  | Contains the SQLite database management classes that an application would use to manage its own private database.  |
| [android.drm](http://code.google.com/android/reference/android/drm/package-summary.html)  |   |
| [android.graphics](http://code.google.com/android/reference/android/graphics/package-summary.html)  | Provides low level graphics tools such as canvases, color filters, points, and rectangles that let you handle drawing to the screen directly.  |
| [android.graphics.drawable](http://code.google.com/android/reference/android/graphics/drawable/package-summary.html)  | Provides classes to manage a variety of visual elements that are intended for display only, such as bitmaps and gradients.  |
| [android.graphics.drawable.shapes](http://code.google.com/android/reference/android/graphics/drawable/shapes/package-summary.html)  |   |
| [android.graphics.glutils](http://code.google.com/android/reference/android/graphics/glutils/package-summary.html)  |   |
| [android.hardware](http://code.google.com/android/reference/android/hardware/package-summary.html)  | Provides support for hardware devices that may not be present on every Android device.  |
| [android.location](http://code.google.com/android/reference/android/location/package-summary.html)  | Classes defining Android location-based and related services.  |
| [android.media](http://code.google.com/android/reference/android/media/package-summary.html)  |   |
| [android.net](http://code.google.com/android/reference/android/net/package-summary.html)  | Classes that help with network access, beyond the normal java.net.\* APIs.  |
| [android.opengl](http://code.google.com/android/reference/android/opengl/package-summary.html)  | Provides OpenGL utilities.  |
| [android.os](http://code.google.com/android/reference/android/os/package-summary.html)  | Provides basic operating system services, message passing, and inter-process communication on the device.  |
| [android.provider](http://code.google.com/android/reference/android/provider/package-summary.html)  | Provides convenience classes to access the content providers supplied by Android.  |
| [android.sax](http://code.google.com/android/reference/android/sax/package-summary.html)  | A framework that makes it easy to write efficient and robust SAX handlers.  |
| [android.telephony](http://code.google.com/android/reference/android/telephony/package-summary.html)  | Provides tools to make, receive, and monitor phone calls and phone status.  |
| [android.telephony.gsm](http://code.google.com/android/reference/android/telephony/gsm/package-summary.html)  | Provides classes to control or read data from GSM phones.  |
| [android.telephony.gsm.stk](http://code.google.com/android/reference/android/telephony/gsm/stk/package-summary.html)  | Provides classes for SIM Toolkit Service.  |
| [android.test.mock](http://code.google.com/android/reference/android/test/mock/package-summary.html)  |   |
| [android.text](http://code.google.com/android/reference/android/text/package-summary.html)  | Provides classes used to render or track text and text spans on the screen.  |
| [android.text.method](http://code.google.com/android/reference/android/text/method/package-summary.html)  | Provides classes that monitor or modify keypad input.  |
| [android.text.style](http://code.google.com/android/reference/android/text/style/package-summary.html)  | Provides classes used to view or change the style of a span of text in a View object.  |
| [android.text.util](http://code.google.com/android/reference/android/text/util/package-summary.html)  |   |
| [android.util](http://code.google.com/android/reference/android/util/package-summary.html)  | Provides common utility methods such as date/time manipulation, base64 encoders and decoders, string and number conversion methods, and XML utilities.  |
| [android.view](http://code.google.com/android/reference/android/view/package-summary.html)  | Provides classes that expose basic user interface classes that handle screen layout and interaction with the user.  |
| [android.view.animation](http://code.google.com/android/reference/android/view/animation/package-summary.html)  | Provides classes that handle tweened animations.  |
| [android.webkit](http://code.google.com/android/reference/android/webkit/package-summary.html)  | Provides tools for browsing the web.  |
| [android.widget](http://code.google.com/android/reference/android/widget/package-summary.html)  | The widget package contains (mostly visual) UI elements to use on your Application screen.  |
| [com.google.android.gtalkservice](http://code.google.com/android/reference/com/google/android/gtalkservice/package-summary.html)  |   |
| [com.google.android.maps](http://code.google.com/android/reference/com/google/android/maps/package-summary.html)  |   |
| [java.io](http://code.google.com/android/reference/java/io/package-summary.html)  | Provides input and output facilities by means of streaming, filesystem access and serialization.  |
| [java.lang](http://code.google.com/android/reference/java/lang/package-summary.html)  | Provides core classes of the Android environment.  |
| [java.lang.annotation](http://code.google.com/android/reference/java/lang/annotation/package-summary.html)  | Provides a couple of pre-defined annotation classes as well as exceptions that can occur when dealing with annotations.  |
| [java.lang.instrument](http://code.google.com/android/reference/java/lang/instrument/package-summary.html)  | Provides classes and interfaces needed for instrumenting applications.  |
| [java.lang.ref](http://code.google.com/android/reference/java/lang/ref/package-summary.html)  |   |
| [java.lang.reflect](http://code.google.com/android/reference/java/lang/reflect/package-summary.html)  |   |
| [java.math](http://code.google.com/android/reference/java/math/package-summary.html)  | This package provides arbitrary-precision integers and decimals.  |
| [java.net](http://code.google.com/android/reference/java/net/package-summary.html)  | Provides networking-related functionality, such as streaming and datagram sockets, handling of Internet addresses, and dealing with HTTP requests.  |
| [java.nio](http://code.google.com/android/reference/java/nio/package-summary.html)  | Provides buffers that help handling data.  |
| [java.nio.channels](http://code.google.com/android/reference/java/nio/channels/package-summary.html)  | Channels provide a way to connect to sources of data such as files, sockets or other structures that allow input and/or output of data.  |
| [java.nio.channels.spi](http://code.google.com/android/reference/java/nio/channels/spi/package-summary.html)  | Service-provider classes for nio channels.  |
| [java.nio.charset](http://code.google.com/android/reference/java/nio/charset/package-summary.html)  | This package allows translating between bytes and different character sets.  |
| [java.nio.charset.spi](http://code.google.com/android/reference/java/nio/charset/spi/package-summary.html)  | Service-provider classe for nio charset.  |
| [java.security](http://code.google.com/android/reference/java/security/package-summary.html)  |   |
| [java.security.acl](http://code.google.com/android/reference/java/security/acl/package-summary.html)  |   |
| [java.security.cert](http://code.google.com/android/reference/java/security/cert/package-summary.html)  |   |
| [java.security.interfaces](http://code.google.com/android/reference/java/security/interfaces/package-summary.html)  |   |
| [java.security.spec](http://code.google.com/android/reference/java/security/spec/package-summary.html)  |   |
| [java.sql](http://code.google.com/android/reference/java/sql/package-summary.html)  | Provides a standard interface for accessing SQL-based databases.  |
| [java.text](http://code.google.com/android/reference/java/text/package-summary.html)  | The java.text package allows to uncouple the text in the application from a natural language.  |
| [java.util](http://code.google.com/android/reference/java/util/package-summary.html)  | Provides an extensive set of utility classes.  |
| [java.util.concurrent](http://code.google.com/android/reference/java/util/concurrent/package-summary.html)  | Utility classes commonly useful in concurrent programming.  |
| [java.util.concurrent.atomic](http://code.google.com/android/reference/java/util/concurrent/atomic/package-summary.html)  | A small toolkit of classes that support lock-free thread-safe programming on single variables.  |
| [java.util.concurrent.locks](http://code.google.com/android/reference/java/util/concurrent/locks/package-summary.html)  | Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors.  |
| [java.util.jar](http://code.google.com/android/reference/java/util/jar/package-summary.html)  | The java.jar package gives access to jar files content.  |
| [java.util.logging](http://code.google.com/android/reference/java/util/logging/package-summary.html)  | This package allows to add logging to any application.  |
| [java.util.prefs](http://code.google.com/android/reference/java/util/prefs/package-summary.html)  | Provides a preferences mechanism, that is, a means of writing configuration data (key/value pairs) to a persistent data store and retrieving it from there.  |
| [java.util.regex](http://code.google.com/android/reference/java/util/regex/package-summary.html)  | Provides an implementation of regular expressions, which is useful for matching, searching, and replacing strings based on patterns.  |
| [java.util.zip](http://code.google.com/android/reference/java/util/zip/package-summary.html)  | This package can read and write the standard zip format.  |
| [javax.crypto](http://code.google.com/android/reference/javax/crypto/package-summary.html)  |   |
| [javax.crypto.interfaces](http://code.google.com/android/reference/javax/crypto/interfaces/package-summary.html)  |   |
| [javax.crypto.spec](http://code.google.com/android/reference/javax/crypto/spec/package-summary.html)  |   |
| [javax.microedition.khronos.opengles](http://code.google.com/android/reference/javax/microedition/khronos/opengles/package-summary.html)  |   |
| [javax.net](http://code.google.com/android/reference/javax/net/package-summary.html)  |   |
| [javax.net.ssl](http://code.google.com/android/reference/javax/net/ssl/package-summary.html)  |   |
| [javax.security.auth](http://code.google.com/android/reference/javax/security/auth/package-summary.html)  |   |
| [javax.security.auth.callback](http://code.google.com/android/reference/javax/security/auth/callback/package-summary.html)  |   |
| [javax.security.auth.login](http://code.google.com/android/reference/javax/security/auth/login/package-summary.html)  |   |
| [javax.security.auth.x500](http://code.google.com/android/reference/javax/security/auth/x500/package-summary.html)  |   |
| [javax.security.cert](http://code.google.com/android/reference/javax/security/cert/package-summary.html)  |   |
| [javax.sound.midi](http://code.google.com/android/reference/javax/sound/midi/package-summary.html)  |   |
| [javax.sound.midi.spi](http://code.google.com/android/reference/javax/sound/midi/spi/package-summary.html)  |   |
| [javax.sound.sampled](http://code.google.com/android/reference/javax/sound/sampled/package-summary.html)  |   |
| [javax.sound.sampled.spi](http://code.google.com/android/reference/javax/sound/sampled/spi/package-summary.html)  |   |
| [javax.sql](http://code.google.com/android/reference/javax/sql/package-summary.html)  | Provides extensions to the standard interface for accessing SQL-based databases.  |
| [javax.xml.parsers](http://code.google.com/android/reference/javax/xml/parsers/package-summary.html)  | Provides facilities for parsing XML documents and building Document Object Model (DOM) trees from them.  |
| [junit.extensions](http://code.google.com/android/reference/junit/extensions/package-summary.html)  |   |
| [junit.framework](http://code.google.com/android/reference/junit/framework/package-summary.html)  |   |
| [org.apache.http](http://code.google.com/android/reference/org/apache/http/package-summary.html)  | The core interfaces and classes of the HTTP components.  |
| [org.apache.http.auth](http://code.google.com/android/reference/org/apache/http/auth/package-summary.html)  | The API for client-side HTTP authentication against a server, commonly referred to as *HttpAuth*.  |
| [org.apache.http.auth.params](http://code.google.com/android/reference/org/apache/http/auth/params/package-summary.html)  | Parameters for configuring *HttpAuth*.  |
| [org.apache.http.client](http://code.google.com/android/reference/org/apache/http/client/package-summary.html)  | The API for client-side HTTP communication and entry point to the *HttpClient* module.  |
| [org.apache.http.client.methods](http://code.google.com/android/reference/org/apache/http/client/methods/package-summary.html)  | Request implementations for the various HTTP methods like GET and POST.  |
| [org.apache.http.client.methods.multipart](http://code.google.com/android/reference/org/apache/http/client/methods/multipart/package-summary.html)  | Provides Multipart support classes for the [MultipartPostMethod](http://code.google.com/android/reference/org/apache/commons/httpclient/methods/MultipartPostMethod.html).  |
| [org.apache.http.client.params](http://code.google.com/android/reference/org/apache/http/client/params/package-summary.html)  | Parameters for configuring *HttpClient*.  |
| [org.apache.http.client.protocol](http://code.google.com/android/reference/org/apache/http/client/protocol/package-summary.html)  | Additional request and response interceptors.  |
| [org.apache.http.client.utils](http://code.google.com/android/reference/org/apache/http/client/utils/package-summary.html)  | Helpers and utility classes for *HttpClient*.  |
| [org.apache.http.conn](http://code.google.com/android/reference/org/apache/http/conn/package-summary.html)  | The client-side connection management and handling API at the heart of what is referred to as *HttpConn*.  |
| [org.apache.http.conn.params](http://code.google.com/android/reference/org/apache/http/conn/params/package-summary.html)  | Parameters for configuring *HttpConn*.  |
| [org.apache.http.conn.ssl](http://code.google.com/android/reference/org/apache/http/conn/ssl/package-summary.html)  | TLS/SSL specific parts of the *HttpConn* API.  |
| [org.apache.http.conn.util](http://code.google.com/android/reference/org/apache/http/conn/util/package-summary.html)  |   |
| [org.apache.http.cookie](http://code.google.com/android/reference/org/apache/http/cookie/package-summary.html)  | The API for client-side state management via cookies, commonly referred to as *HttpCookie*.  |
| [org.apache.http.cookie.params](http://code.google.com/android/reference/org/apache/http/cookie/params/package-summary.html)  | Parameters for configuring *HttpCookie*.  |
| [org.apache.http.entity](http://code.google.com/android/reference/org/apache/http/entity/package-summary.html)  | Representations for HTTP message entities.  |
| [org.apache.http.impl](http://code.google.com/android/reference/org/apache/http/impl/package-summary.html)  | Default implementations for interfaces in [org.apache.http](http://code.google.com/android/reference/org/apache/http/package-summary.html).  |
| [org.apache.http.impl.auth](http://code.google.com/android/reference/org/apache/http/impl/auth/package-summary.html)  |   |
| [org.apache.http.impl.client](http://code.google.com/android/reference/org/apache/http/impl/client/package-summary.html)  |   |
| [org.apache.http.impl.conn](http://code.google.com/android/reference/org/apache/http/impl/conn/package-summary.html)  |   |
| [org.apache.http.impl.conn.tsccm](http://code.google.com/android/reference/org/apache/http/impl/conn/tsccm/package-summary.html)  | The implementation of a thread-safe client connection manager.  |
| [org.apache.http.impl.cookie](http://code.google.com/android/reference/org/apache/http/impl/cookie/package-summary.html)  |   |
| [org.apache.http.impl.entity](http://code.google.com/android/reference/org/apache/http/impl/entity/package-summary.html)  | Default implementations for interfaces in [org.apache.http.entity](http://code.google.com/android/reference/org/apache/http/entity/package-summary.html).  |
| [org.apache.http.impl.io](http://code.google.com/android/reference/org/apache/http/impl/io/package-summary.html)  | Default implementations for interfaces in [org.apache.http.io](http://code.google.com/android/reference/org/apache/http/io/package-summary.html).  |
| [org.apache.http.io](http://code.google.com/android/reference/org/apache/http/io/package-summary.html)  | The transport layer abstraction of the HTTP components.  |
| [org.apache.http.message](http://code.google.com/android/reference/org/apache/http/message/package-summary.html)  | A selection of HTTP [message](http://code.google.com/android/reference/org/apache/http/message/AbstractHttpMessage.html) implementations.  |
| [org.apache.http.params](http://code.google.com/android/reference/org/apache/http/params/package-summary.html)  | The parameterization framework for HTTP components.  |
| [org.apache.http.protocol](http://code.google.com/android/reference/org/apache/http/protocol/package-summary.html)  | HTTP protocol execution framework.  |
| [org.apache.http.util](http://code.google.com/android/reference/org/apache/http/util/package-summary.html)  | Mostly utility classes with static helper methods for various purposes.  |
| [org.apache.http.util.concurrent](http://code.google.com/android/reference/org/apache/http/util/concurrent/package-summary.html)  | Interfaces corresponding to java.util.concurrent.  |
| [org.bluez](http://code.google.com/android/reference/org/bluez/package-summary.html)  | Provides classes to manage Bluetooth functionality on the device.  |
| [org.json](http://code.google.com/android/reference/org/json/package-summary.html)  |   |
| [org.w3c.dom](http://code.google.com/android/reference/org/w3c/dom/package-summary.html)  | Provides the official W3C Java bindings for the Document Object Model, level 2 core.  |
| [org.xml.sax](http://code.google.com/android/reference/org/xml/sax/package-summary.html)  | This package provides the core SAX APIs.  |
| [org.xml.sax.ext](http://code.google.com/android/reference/org/xml/sax/ext/package-summary.html)  | This package contains interfaces to SAX2 facilities that conformant SAX drivers won't necessarily support.  |
| [org.xml.sax.helpers](http://code.google.com/android/reference/org/xml/sax/helpers/package-summary.html)  | This package contains "helper" classes, including support for bootstrapping SAX-based applications.  |