Android

Android FAQs

Answers to common questions can be found in the following topic areas:

General questions about Android

Overview questions about developing apps on Android

Common Tasks and How To Do Them

Frequently performed tasks in developing Android applications

Troubleshooting Tips

Answers to troubleshooting common problems

Open Source Licensing

Common topics around licensing and Android Open Source

Application Framework

Common questions about the Android Application Framework

Security

Answers to common security questions

Build 110632-110632 - 22 Sep 2008 13:34

Android

General Android

What languages does Android support?

Android applications are written using the Java programming language.

Can I write code for Android using C/C++?

Android only supports applications written using the Java programming language at this time.

Will Android run on *<insert phone here>*?

No. There is currently only an Android SDK for the Windows, Mac OS X(intel), and Linux(i386) platforms. The SDK includes a software emulator on which you can test Android applications.

I live in *insert country here*. Am I eligible for the Android Developers Challenge?

Please see: http://code.google.com/android/adc.html and http://code.google.com/android/adc_faq.html for more details about the Android Developers Challenge.

Build 110632-110632 - 22 Sep 2008 13:34

Android

Common Tasks and How To Do Them in Android

- Creating an Android Application using the Eclipse plugin
- Creating an Android Application without the Eclipse plugin
- Adding an External Library (.jar) using Eclipse
- Implementing Activity callbacks (Android calls your activity at various key moments in its life cycle. You must know how to handle each of these to draw your screen, initialize class members, and acquire data.)
- Opening a new screen
- Listening for button clicks
- Configuring general window properties
- Referring to localhost from the emulated environment
- Storing and retrieving state
- Storing and retrieving preferences
- Storing and retrieving larger or more complex persistent data (files and data)
- Playing audio, video, still, or other media files
- Listening for and broadcasting global messages and setting alarms
- Displaying alerts
- Displaying a progress bar
- Adding items to the screen menu
- Display a web page
- Binding to data
- Capture images from the phone camera
- Handling expensive operations in the UI thread
- Selecting, highlighting, or styling portions of text
- Utilizing attributes in a Map query
- List of files for an Android application
- Print messages to a log file

The ApiDemos sample application includes many, many examples of common tasks and UI features. See the code inside samples/ApiDemos and the other sample applications under the samples/ folder in the SDK.

Creating an Android Application using the Eclipse Plugin

Using the Android Eclipse plugin is the fastest and easiest way to start creating a new Android application. The plugin automatically generates the correct project structure for your application, and keeps the resources compiled for you automatically.

It is still a good idea to know what is going on though. Take a look at Overview of an Android Application to understand the basics of how an Android application works.

It is also recommended that you take a look at the ApiDemos application and the other sample applications in the samples/ folder in the SDK.

Finally, a great way to started with Android development in Eclipse is to follow both the Hello Android and Notepad code tutorials. In particular, the start of the Hello Android tutorial is an excellent introduction to creating a new Android application in Eclipse.

Creating an Android Application without the Eclipse Plugin

This topic describes the manual steps in creating an Android application. Before reading this, you should read Overview of an

Android Application to understand the basics of how an Android application works. You might also want to look at the sample applications that ship with Android under the samples/ directory.

Here is a list of the basic steps in building an application.

1. **Create your required resource files** This includes the AndroidManifest.xml global description file, string files that your application needs, and layout files describing your user interface. A full list of optional and required files and syntax details for each is given in File List for an Android Application.
2. **Design your user interface** See Implementing a UI for details on elements of the Android screen.
3. **Implement your Activity** (this page) You will create one class/file for each screen in your application. Screens will inherit from an android.app class, typically android.app.Activity for basic screens, android.app.ListActivity for list screens, or android.app.Dialog for dialog boxes. You will implement the required callbacks that let you draw your screen, query data, and commit changes, and also perform any required tasks such as opening additional screens or reading data from the device. Common tasks, such as opening a new screen or reading data from the device, are described below. The list of files you'll need for your application are described in List of Files for an Android Application.
4. **Build and install your package.** The Android SDK has some nice tools for generating projects and debugging code.

Adding an External Library (.jar) using Eclipse

You can use a third party JAR in your application by adding it to your Eclipse project as follows:

1. In the **Package Explorer** panel, right-click on your project and select **Properties**.
2. Select **Java Build Path**, then the tab **Libraries**.
3. Press the **Add External JARs...** button and select the JAR file.

Alternatively, if you want to include third party JARs with your package, create a new directory for them within your project and select **Add Library...** instead.

It is not necessary to put external JARs in the assets folder.

Implementing Activity Callbacks

Android calls a number of callbacks to let you draw your screen, store data before pausing, and refresh data after closing. You must implement at least some of these methods. See Lifetime of a Screen to learn when and in what order these methods are called. Here are some of the standard types of screen classes that Android provides:

- android.app.Activity - This is a standard screen, with no specialization.
- android.app.ListActivity - This is a screen that is used to display a list of something. It hosts a ListView object, and exposes methods to let you identify the selected item, receive callbacks when the selected item changes, and perform other list-related actions.
- android.app.Dialog - This is a small, popup dialog-style window that isn't intended to remain in the history stack. (It is not resizeable or moveable by the user.)

Opening a New Screen

Your Activity will often need to open another Activity screen as it progresses. This new screen can be part of the same application or part of another application, the new screen can be floating or full screen, it can return a result, and you can decide whether to close this screen and remove it from the history stack when you are done with it, or to keep the screen open in history. These next sections describe all these options.

Floating or full?

When you open a new screen you can decide whether to make it transparent or floating, or full-screen. The choice of new screen affects the event sequence of events in the old screen (if the new screen obscures the old screen, a different series of events is called in the old screen). See Lifetime of an Activity for details.

Transparent or floating windows are implemented in three standard ways:

- Create an app.Dialog class
- Create an app.AlertDialog class

- Set the Theme_Dialog *theme* attribute to `@android:style/Theme.Dialog` in your AndroidManifest.xml file. For example:

```
<activity class="AddRssItem" android:label="Add an item"
android:theme="@android:style/Theme.Dialog"/>
```

Calling startActivity() or startActivityForResult() will open a new screen in whatever way it defines itself (if it uses a floating theme it will be floating, otherwise it will be full screen).

Opening a Screen

When you want to open a new screen, you can either explicitly specify the activity class to open, or you can let the operating system decide which screen to open, based upon the data and various parameters you pass in. A screen is opened by calling startActivity and passing in an Intent object, which specifies the criteria for the handling screen. To specify a specific screen, call Intent.setClass or setClassName with the exact activity class to open. Otherwise, set a variety of values and data, and let Android decide which screen is appropriate to open. Android will find one or zero Activities that match the specified requirements; it will never open multiple activities for a single request. More information on Intents and how Android resolves them to a specific class is given in the Intent topic.

Some Intent examples

The following snippet loads the com.android.samples.Animation1 class, and passes it some arbitrary data.:

```
Intent myIntent = new Intent();
myIntent.setClassName("com.android.samples", "com.android.samples.Animation1");
myIntent.putExtra("com.android.samples.SpecialValue", "Hello, Joe!"); // key/value pair, where key needs
current package prefix.
startActivity(myIntent);
```

The next snippet requests that a Web page be opened by specifying the VIEW action, and a URI data string starting with "http://" schema:

```
Intent myIntent = new Intent(Intent.VIEW_ACTION, Uri.parse("http://www.google.com"));
```

Here is the intent filter from the AndroidManifest.xml file for com.android.browser:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <scheme android:name="http" />
    <scheme android:name="https" />
    <scheme android:name="file" />
</intent-filter>
```

Android defines a number of standard values, for instance the action constants defined by Intent. You can define custom values, but both the caller and handler must use them. See the <intent-filter> tag description in AndroidManifest.xml File Details for more information on the manifest syntax for the handling application.

Returning a Result from a Screen

A window can return a result after it closes. This result will be passed back into the calling Activity's onActivityResult() method, which can supply an Intent containing arbitrary data, along with the request code passed to startActivityForResult(). Note that you must call the startActivityForResult() method that accepts a request code parameter to get this callback. The following code demonstrates opening a new screen and retrieving a result.

```
// Open the new screen.
public void onClick(View v){
    // Start the activity whose result we want to retrieve.  The
    // result will come back with request code GET_CODE.
    Intent intent = new Intent(this, com.example.app.ChooseYourBoxer.class);
    startActivityForResult(intent, CHOOSE_FIGHTER);
```

```
}

// Listen for results.
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    // See which child activity is calling us back.
    switch (resultCode) {
        case CHOOSE_FIGHTER:
            // This is the standard resultCode that is sent back if the
            // activity crashed or didn't doesn't supply an explicit result.
            if (resultCode == RESULT_CANCELED){
                myMessageboxFunction("Fight cancelled");
            }
            else {
                myFightFunction(data);
            }
        default:
            break;
    }
}

// Class SentResult
// Temporary screen to let the user choose something.
    private OnClickListener mLincolnListener = new OnClickListener(){
        public void onClick(View v) {
            Bundle stats = new Bundle();
            stats.putString("height","6\'4\"");
            stats.putString("weight", "190 lbs");
            stats.putString("reach", "74\"");
            setResult(RESULT_OK, "Lincoln", stats);
            finish();
        }
    };

    private OnClickListener mWashingtonListener = new OnClickListener() {
        public void onClick(View v){
            Bundle stats = new Bundle();
            stats.putString("height","6\'2\"");
            stats.putString("weight", "190 lbs");
            stats.putString("reach", "73\"");
            setResult(RESULT_OK, "Washington", Bundle);
            finish();
        }
    };
```

Lifetime of the new screen

An activity can remove itself from the history stack by calling Activity.finish() on itself, or the activity that opened the screen can call Activity.finishActivity() on any screens that it opens to close them.

Listening for Button Clicks

Button click and other UI event capturing are covered in Listening for UI Notifications on the UI Design page.

Configuring General Window Properties

You can set a number of general window properties, such as whether to display a title, whether the window is floating, and whether it displays an icon, by calling methods on the Window member of the underlying View object for the window. Examples include calling getWindow().requestFeature() (or the convenience method requestWindowFeature(*some_feature*)) to hide the title. Here is an example of hiding the title bar:

```
//Hide the title bar
requestWindowFeature(Window.FEATURE_NO_TITLE);
```

Referring to localhost from the emulated environment

If you need to refer to your host computer's *localhost*, such as when you want the emulator client to contact a server running on the same host, use the alias `10.0.2.2` to refer to the host computer's loopback interface. From the emulator's perspective, localhost (`127.0.0.1`) refers to its own loopback interface.

Storing and Retrieving State

If your application is dumped from memory because of space concerns, it will lose all user interface state information such as checkbox state and text box values as well as class member values. Android calls Activity.onSaveInstanceState before it pauses the application. This method hands in a Bundle that can be used to store name/value pairs that will persist and be handed back to the application even if it is dropped from memory. Android will pass this Bundle back to you when it calls onCreate(). This Bundle only exists while the application is still in the history stack (whether or not it has been removed from memory) and will be lost when the application is finalized. See the topics for onSaveInstanceState(Bundle) and onCreate(Bundle) for examples of storing and retrieving state.

Read more about the life cycle of an application in Lifetime of an Activity.

Storing and Retrieving Larger or More Complex Persistent Data

Your application can store files or complex collection objects, and reserve them for private use by itself or other activities in the application, or it can expose its data to all other applications on the device. See Storing, Retrieving, and Exposing Data to learn how to store and retrieve private data, how to store and retrieve common data from the device, and how to expose your private data to other applications.

Playing Media Files

Please see the document Android Media APIs for more details.

Listening For and Broadcasting Global Messages, and Setting Alarms

You can create a listening class that can be notified or even instantiated whenever a specific type of system message is sent.

The listening classes, called broadcast receivers, extend BroadcastReceiver. If you want Android to instantiate the object whenever an appropriate intent notification is sent, define the receiver with a `<receiver>` element in the AndroidManifext.xml file. If the caller is expected to instantiate the object in preparation to receive a message, this is not required. The receiver will get a call to their BroadcastReceiver.onReceive() method. A receiver can define an `<intent-filter>` tag that describes the types of messages it will receive. Just as Android's IntentResolver will look for appropriate Activity matches for a startActivity() call, it will look for any matching Receivers (but it will send the message to all matching receiver, not the "best" match).

To send a notification, the caller creates an Intent object and calls Context.sendBroadcast() with that Intent. Multiple recipients can receive the same message. You can broadcast an Intent message to an intent receiver in any application, not only your own. If the receiving class is not registered using `<receiver>` in its manifest, you can dynamically instantiate and register a receiver by calling Context.registerReceiver().

Receivers can include intent filters to specify what kinds of intents they are listening for. Alternatively, if you expect a single known caller to contact a single known receiver, the receiver does not specify an intent filter, and the caller specifies the receiver's class name in the Intent by calling Intent.setClassName() with the recipient's class name. The recipient receives a Context object that refers to its own package, not to the package of the sender.

Note: If a receiver or broadcaster enforces permissions, your application might need to request permission to send or receive messages from that object. You can request permission by using the <uses-permission> tag in the manifest.

Here is a code snippet of a sender and receiver. This example does not demonstrate registering receivers dynamically. For a full code example, see the AlarmService class in the ApiDemos project.

Sending the message

```
// We are sending this to a specific recipient, so we will
// only specify the recipient class name.
Intent intent = new Intent(this, AlarmReceiver.class);
intent.putExtra("message","Wake up.");
sendBroadcast(intent);
```

Receiving the message

Receiver AndroidManifest.xml (because there is no intent filter child, this class will only receive a broadcast when the receiver class is specified by name, as is done in this example):

```
<receiver class=".AlarmReceiver" />
```

Receiver Java code:

```
public class AlarmReceiver extends BroadcastReceiver{
    // Display an alert that we've received a message.
    @Override
    public void onReceive(Context context, Intent intent){
            // Send a text notification to the screen.
        NotificationManager nm = (NotificationManager)
        context.getSystemService(Context.NOTIFICATION_SERVICE);
        nm.notifyWithText(R.id.alarm,
                          "Alarm!!!",
                          NotificationManager.LENGTH_SHORT,
                          null);
   }
}
```

Other system messages

You can listen for other system messages sent by Android as well, such as USB connection/removal messages, SMS arrival messages, and timezone changes. See Intent for a list of broadcast messages to listen for. Messages are marked "Broadcast Action" in the documentation.

Listening for phone events

The android.telephony package overview page describes how to register to listen for phone events.

Setting Alarms

Android provides an AlarmManager service that will let you specify an Intent to send at a designated time. This intent is typically used to start an application at a preset time. (Note: If you want to send a notification to a sleeping or running application, use Handler instead.)

Displaying Alerts

There are two major kinds of alerts that you may display to the user: (1) Normal alerts are displayed in response to a user action, such as trying to perform an action that is not allowed. (2) Out-of-band alerts, called notifications, are displayed as a result of something happening in the background, such as the user receiving new e-mail.

Normal Alerts

Android provides a number of ways for you to show popup notifications to your user as they interact with your application.

Class	Description

app.Dialog	A generic floating dialog box with a layout that you design.
app.AlertDialog	A popup alert dialog with two buttons (typically OK and Cancel) that take callback handlers. See the section after this table for more details.
ProgressDialog	A dialog box used to indicate progress of an operation with a known progress value or an indeterminate length (setProgress(bool)). See **Views** > **Progress Bar** in ApiDemos for examples.
Activity	By setting the theme of an activity to android:theme="android:style/Theme.Dialog", your activity will take on the appearance of a normal dialog, floating on top of whatever was underneath it. You usually set the theme through the android:theme attribute in your AndroidManifest.xml. The advantage of this over Dialog and AlertDialog is that Application has a much better managed life cycle than dialogs: if a dialog goes to the background and is killed, you cannot recapture state, whereas Application exposes a Bundle of saved values in `onCreate()` to help you maintain state.

AlertDialog

This is a basic warning dialog box that lets you configure a message, button text, and callback. You can create one by calling using the AlertDialog.Builder class, as shown here.

```
private Handler mHandler = new Handler() {
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case ACCEPT_CALL:
            answer(msg.obj);
            break;

            case BOUNCE_TO_VOICEMAIL:
            voicemail(msg.obj);
            break;

        }
    }
};


private void IncomingMotherInlawCall(Connection c) {
    String Text;

    // "Answer" callback.
    Message acceptMsg = Message.obtain();
    acceptMsg.target = mHandler;
    acceptMsg.what = ACCEPT_CALL;
    acceptMsg.obj = c.getCall();

    // "Cancel" callback.
    final Message rejectMsg = Message.obtain();
    rejectMsg.target = mHandler;
    rejectMsg.what = BOUNCE_TO_VOICEMAIL;
    rejectMsg.obj = c.getCall();

    new AlertDialog.Builder(this)
      .setMessage("Phyllis is calling")
      .setPositiveButton("Answer", acceptMsg)
      .setOnCanceListener(new OnCancelListener() {
        public void onCancel(DialogInterface dialog) {
          rejectMsg.sendToTarget();
        }});
      .show();
}
```

Notifications

Out-of-band alerts should always be displayed using the NotificationManager, which allows you to tell the user about

something they may be interested in without disrupting what they are currently doing. A notification can be anything from a brief pop-up box informing the user of the new information, through displaying a persistent icon in the status bar, to vibrating, playing sounds, or flashing lights to get the user's attention. In all cases, the user must explicitly shift their focus to the notification before they can interact with it.

The following code demonstrates using NotificationManager to display a basic text popup when a new SMS message arrives in a listening service, and provides the current message count. You can see several more examples in the ApiDemos application, under app/ (named *notification**.java).

```
static void setNewMessageIndicator(Context context, int messageCount){
   // Get the static global NotificationManager object.
   NotificationManager nm = NotificationManager.getDefault();



   // If we're being called because a new message has been received,
   // then display an icon and a count. Otherwise, delete the persistent
   // message.
   if (messageCount > 0) {
      nm.notifyWithText(myApp.NOTIFICATION_GUID,      // ID for this notification.
                messageCount + " new message" + messageCount > 1 ? "s":"", // Text to display.
                NotificationManager.LENGTH_SHORT); // Show it for a short time only.
   }
}
```

To display a notification in the status bar and have it launch an intent when the user selects it (such as the new text message notification does), call NotificationManager.notify(), and pass in vibration patterns, status bar icons, or Intents to associate with the notification.

Displaying a Progress Bar

An activity can display a progress bar to notify the user that something is happening. To display a progress bar in a screen, call Activity.requestWindowFeature(Window.FEATURE_PROGRESS). To set the value of the progress bar, call Activity.getWindow().setFeatureInt(Window.FEATURE_PROGRESS, *level*). Progress bar values are from 0 to 9,999, or set the value to 10,000 to make the progress bar invisible.

You can also use the ProgressDialog class, which enables a dialog box with an embedded progress bar to send a "I'm working on it" notification to the user.

Adding Items to the Screen Menu

Every Android screen has a default menu with default options, such as adding the activity to the favorites menu. You can add your own menu entries to the default menu options by implementing Activity.onCreateOptionsMenu or Activity.onPrepareOptionsMenu(), and adding Item objects to the Menu passed in. To handle clicks implement Activity.onOptionsItemSelected() to handle the click in your Activity class. You may also pass the Item object a handler class that implements the Runnable class (a handler) but this is less efficient and discouraged.

An application receives a callback at startup time to enable it to populate its menu. Additionally, it receives callbacks each time the user displays the options menu to let you perform some contextual modifications on the menu. To populate the menu on startup, override Activity.onCreateOptionsMenu; to populate it when the menu is called (somewhat less efficient), you can override Activity.onPrepareOptionsMenu(). Each Activity has its own menu list.

Menu items are displayed in the order added, though you can group them as described in the Menu.add documentation. The following code snippet adds three items to the default menu options and handles them through the overridden Activity.onOptionsItemSelected() method. You can show or hide menu items by calling setVisible() or setGroupVisible().

```
// Called only the first time the options menu is displayed.
// Create the menu entries.
// Menu adds items in the order shown.
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
```

```
    // Parameters for menu.add are:
    // group -- Not used here.
    // id -- Used only when you want to handle and identify the click yourself.
    // title
    menu.add(0, 0, "Zoom");
    menu.add(0, 1, "Settings");
    menu.add(0, 2, "Other");
    return true;
}

// Activity callback that lets your handle the selection in the class.
// Return true to indicate that you've got it, false to indicate
// that it should be handled by a declared handler object for that
// item (handler objects are discouraged for reasons of efficiency).
@Override
public boolean onOptionsItemSelected(MenuItem item){
    switch (item.getId()) {
    case 0:
        showAlert("Menu Item Clicked", "Zoom", "ok", null, false, null);
        return true;
    case 1:
        showAlert("Menu Item Clicked", "Settings", "ok", null, false, null);
        return true;
    case 2:
        showAlert("Menu Item Clicked", "Other", "ok", null, false, null);
        return true;
    }
    return false;
}
```

You can add key shortcuts by calling the MenuItem.setAlphabeticShortcut() or MenuItem.setNumericShortcut() methods, as demonstrated here to add a "c" shortcut to a menu item:

```
thisItem.setAlphabeticShortcut('c');
```

Adding Submenus

Add a submenu by calling Menu.addSubMenu(), which returns a SubMenu object. You can then add additional items to this menu. Menus can only be one level deep, and you can customize the appearance of the submenu menu item.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    // Parameters for menu.add are:
    // group -- Not used here.
    // id -- Used only when you want to handle and identify the click yourself.
    // title
    menu.add(0, 0, "Send message");
    menu.add(0, 1, "Settings");
    menu.add(0, 2, "Local handler");
    menu.add(0, 3, "Launch contact picker");

    // Add our submenu.
    SubMenu sub = menu.addSubMenu(1, 4, "Days of the week");
    sub.add(0, 5, "Monday");
    sub.add(0, 6, "Tuesday");
    sub.add(0, 7, "Wednesday");
    sub.add(0, 8, "Thursday");
    sub.add(0, 9, "Friday");
    sub.add(0, 10, "Saturday");
    sub.add(0, 11, "Sunday");
    return true;
}
```

Adding yourself to menus on other applications

You can also advertise your Activity's services so that other Activities can add your activity to their own option menu. For example, suppose you implement a new image handling tool that shrinks an image to a smaller size and you would like to offer this as a menu option to any other Activity that handles pictures. To do this, you would exposes your capabilities inside an intent filter in your manifest. If another application that handles photos asks Android for any Activities that can perform actions on pictures, Android will perform intent resolution, find your Activity, and add it to the other Activity's options menu.

The offering application

The application offering the service must include an `<intent-filter>` element in the manifest, inside the `<activity>` tag of the offering Activity. The intent filter includes all the details describing what it can do, such as a `<type>` element that describes the MIME type of data that it can handle, a custom `<action>` value that describes what your handling application can do (this is so that when it receives the Intent on opening it knows what it is expected to do), and most important, include a `<category>` filter with the value `android.intent.category.ALTERNATIVE` and/or `android.intent.category.SELECTED_ALTERNATIVE` (SELECTED_ALTERNATIVE is used to handle only the currently selected element on the screen, rather than the whole Activity intent.

Here's an example of a snip of a manifest that advertises picture shrinking technology for both selected items and the whole screen.

```
<activity class="PictureShrink">                  <!-- Handling class -->
    <intent-filter label="Shrink picture">      <!-- Menu label to display -->
        <action android:name="com.example.sampleapp.SHRINK_IT" />
        <data android:name="image/*" />                  <!-- MIME type for generic images -->
        <category android:name="android.intent.category.ALTERNATIVE " />
        <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    </intent-filter>
</activity>
```

The menu-displaying application

An application that wants to display a menu that includes any additional external services must, first of all, handle its menu creation callback. As part of that callback it creates an intent with the category Intent.ALTERNATIVE_CATEGORY and/or Intent.SELECTED_ALTERNATIVE, the MIME type currently selected, and any other requirements, the same way as it would satisfy an intent filter to open a new Activity. It then calls menu.addIntentOptions() to have Android search for and add any services meeting those requirements. It can optionally add additional custom menu items of its own.

You should implement SELECTED_ALTERNATIVE in onPrepareOptionsMenu() rather than onCreateOptionsMenu(), because the user's selection can change after the application is launched.

Here's a code snippet demonstrating how a picture application would search for additional services to display on its menu.

```
@Override public boolean
onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);

    // Create an Intent that describes the requirements to fulfill to be included
    // in our menu. The offering app must include a category value of Intent.ALTERNATIVE_CATEGORY.
    Intent intent = new Intent(null, getIntent().getData());
    intent.addCategory(Intent.ALTERNATIVE_CATEGORY);

    // Search for, and populate the menu with, acceptable offering applications.
    menu.addIntentOptions(
         0,      // Group
         0,      // Any unique IDs we might care to add.
         MySampleClass.class.getName(),  // Name of the class displaying the menu--here, its this class.
         null,   // No specifics.
         intent, // Previously created intent that describes our requirements.
         0,      // No flags.
         null);  // No specifics.

    return true;
}
```

Display a Web Page

Use the webkit.WebView object.

Binding to Data

You can bind a ListView to a set of underlying data by using a shim class called ListAdapter (or a subclass). ListAdapter subclasses bind to a variety of data sources, and expose a common set of methods such as getItem() and getView(), and uses them to pick View items to display in its list. You can extend ListAdapter and override getView() to create your own custom list items. There are essentially only two steps you need to perform to bind to data:

1. Create a ListAdapter object and specify its data source
2. Give the ListAdapter to your ListView object.

That's it!

Here's an example of binding a ListActivity screen to the results from a cursor query. (Note that the setListAdapter() method shown is a convenience method that gets the page's ListView object and calls setAdapter() on it.)

```
// Run a query and get a Cursor pointing to the results.
Cursor c = People.query(this.getContentResolver(), null);
startManagingCursor(c);

// Create the ListAdapter. A SimpleCursorAdapter lets you specify two interesting things:
// an XML template for your list item, and
// The column to map to a specific item, by ID, in your template.
ListAdapter adapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_1,  // Use a template that displays a text view
                c,                                      // Give the cursor to the list adapter
                new String[] {People.NAME} ,           // Map the NAME column in the people database to...
                new String[] {"text1"});               // The "text1" view defined in the XML template
setListAdapter(adapter);
```

See view/List4 in the ApiDemos project for an example of extending ListAdapter for a new data type.

Capture Images from the Phone Camera

You can hook into the device's camera onto your own Canvas object by using the Camera class. See that class's documentation, and the ApiDemos project's Camera Preview application (Graphics/Camera Preview) for example code.

Handling Expensive Operations in the UI Thread

Avoid performing long-running operations (such as network I/O) directly in the UI thread — the main thread of an application where the UI is run — or your application may be blocked and become unresponsive. Here is a brief summary of the recommended approach for handling expensive operations:

1. Create a Handler object in your UI thread
2. Spawn off worker threads to perform any required expensive operations
3. Post results from a worker thread back to the UI thread's handler either through a Runnable or a Message
4. Update the views on the UI thread as needed

The following outline illustrates a typical implementation:

```
public class MyActivity extends Activity {

    [ . . . ]
    // Need handler for callbacks to the UI thread
    final Handler mHandler = new Handler();

    // Create runnable for posting
    final Runnable mUpdateResults = new Runnable() {
        public void run() {
```

```
            updateResultsInUi();
        }
    };

    @Override
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        [ . . . ]
    }

    protected void startLongRunningOperation() {

        // Fire off a thread to do some work that we shouldn't do directly in the UI thread
        Thread t = new Thread() {
            public void run() {
                mResults = doSomethingExpensive();
                mHandler.post(mUpdateResults);
            }
        };
        t.start();
    }

    private void updateResultsInUi() {

        // Back in the UI thread -- update our UI elements based on the data in mResults
        [ . . . ]
    }
}
```

For further discussions on this topic, see Developing Responsive Applications and the Handler documentation.

Selecting, Highlighting, or Styling Portions of Text

You can highlight or style the formatting of strings or substrings of text in a TextView object. There are two ways to do this:

- If you use a string resource, you can add some simple styling, such as bold or italic using HTML notation. The currently supported tags are: `B` (bold), `I` (italic), `U` (underline), `TT` (monospace), `BIG`, `SMALL`, `SUP` (superscript), `SUB` (subscript), and `STRIKE` (strikethrough). So, for example, in res/values/strings.xml you could declare this:
  ```
  <resource>
      <string>id="@+id/styled_welcome_message">We are <b><i>so</i></b> glad to see you.</string>
  </resources>
  ```
- To style text on the fly, or to add highlighting or more complex styling, you must use the Spannable object as described next.

To style text on the fly, you must make sure the TextView is using Spannable storage for the text (this will always be true if the TextView is an EditText), retrieve its text with getText(), and call setSpan(Object, int, int, int), passing in a new style class from the android.text.style package and the selection range.

The following code snippet demonstrates creating a string with a highlighted section, italic section, and bold section, and adding it to an EditText object.

```
// Get our EditText object.
EditText vw = (EditText)findViewById(R.id.text);

// Set the EditText's text.
vw.setText("Italic, highlighted, bold.");

// If this were just a TextView, we could do:
// vw.setText("Italic, highlighted, bold.", TextView.BufferType.SPANNABLE);
// to force it to use Spannable storage so styles can be attached.
// Or we could specify that in the XML.

// Get the EditText's internal text storage
Spannable str = vw.getText();

// Create our span sections, and assign a format to each.
str.setSpan(new StyleSpan(android.graphics.Typeface.ITALIC), 0, 7, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

```
str.setSpan(new BackgroundColorSpan(0xFFFFFF00), 8, 19, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
str.setSpan(new StyleSpan(android.graphics.Typeface.BOLD), 21, str.length() - 1,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

Utilizing attributes in a Map query

When using a search intent to ask the Maps activity to search for something, the Maps activity responds to the following attributes in the optional context bundle:

```
float "centerLatitude" default 0.0f
float "centerLongitude" default 0.0f
float "latitudeSpan" default 0.0f
float "longitudeSpan" default 0.0f
int "zoomLevel" default 10
```

This context information is used to center the search result in a particular area, and is equivalent to adjusting the Map activity to the described location and zoom level before issuing the query.

If the latitudeSpan, longitudeSpan, and zoomLevel attributes are not consistent, then it is undefined which one takes precedence.

List of Files for an Android Application

The following list describes the structure and files of an Android application. Many of these files can be built for you (or stubbed out) by the activitycreator.py application shipped in the tools/ menu of the SDK. See Building an Android Sample Application for more information on using activitycreator.py.

MyApp/	
AndroidManifest.xml	(*required*) Advertises the screens that this application provides, where they can be launched (from the main program menu or elsewhere), any content providers it implements and what kind of data they handle, where the implementation classes are, and other application-wide information. Syntax details for this file are described in AndroidManifest.xml.
src/ /*myPackagePath*/.../*MyClass*.java	(*required*) This folder holds all the source code files for your application, inside the appropriate package subfolders.
res/	(*required*) This folder holds all the *resources* for your application. Resources are external data files or description files that are compiled into your code at build time. Files in different folders are compiled differently, so you must put the proper resource into the proper folder. (See Resources for details.)
anim/ *animation1*.xml ...	(*optional*) Holds any animation XML description files that the application uses. The format of these files is described in Resources.
drawable/ *some_picture*.png *some_stretchable*.9.png *some_background*.xml ...	(*optional*) Zero or more files that will be compiled to android.graphics.drawable resources. Files can be image files (png, gif, or other) or XML files describing other graphics such as bitmaps, stretchable bitmaps, or gradients. Supported bitmap file formats are PNG (preferred), JPG, and GIF (discouraged), as well as the custom 9-patch stretchable bitmap format. These formats are described in Resources.
layout/	(*optional*) Holds all the XML files describing screens or parts of screens.

screen_1_layout.xml ...	Although you could create a screen in Java, defining them in XML files is typically easier. A layout file is similar in concept to an HTML file that describes the screen layout and components. See Implementing a UI for more information about designing screens, and Layout Resources for the syntax of these files.
values/ arrays classes.xml colors.xml dimens.xml strings.xml styles.xml values.xml	(*optional*) XML files describing additional resources such as strings, colors, and styles. The naming, quantity, and number of these files are not enforced--any XML file is compiled, but these are the standard names given to these files. However, the syntax of these files is prescribed by Android, and described in Resources.
xml/	(*optional*) XML files that can be read at run time on the device.
raw/	(*optional*) Any files to be copied directly to the device.

Print Messages to a Log File

To write log messages from your application:

1. Import `android.util.Log`.
2. Use `Log.v()`, `Log.d()`, `Log.i()`, `Log.w()`, or `Log.e()` to log messages. (See the Log class.)
   E.g., `Log.e(this.toString(), "error: " + err.toString())`
3. Launch DDMS from a terminal by executing `ddms` in your Android SDK `/tools` path.
4. Run your application in the Android emulator.
5. From the DDMS application, select the emulator (e.g., "emulator-5554") and click **Device > Run logcat...** to view all the log data.

Note: If you are running Eclipse and encounter a warning about the VM debug port when opening DDMS, you can ignore it if you're only interested in logs. However, if you want to further inspect and control your processes from DDMS, then you should close Eclipse before launching DDMS so that it may use the VM debugging port.

Android

Troubleshooting

Here are some tips and tricks for common Android errors. Don't forget to use the ddms logcat capability to get a deeper view when errors occur. See Debugging an Android Application for more debugging tips.

ADT Installation Error: "requires plug-in org.eclipse.wst.sse.ui".

The "Android Editors" feature of the ADT Plugin requires specific Eclipse components, such as WST. If you encounter this error message during ADT installation, you need to install the required Eclipse components and then try the ADT installation again. The easiest way to install the required components for the Android Editors feature of ADT is the following:

- From the dialog where you select the **Update sites to visit**, select the checkboxes for both the ADT site, and the Callisto/Europa/Ganymede Discovery Site (you may want to check **Automatically select mirrors** at the bottom).
- Click **Finish**.
- In the **Next** dialog, select the Android Plugins.
- Now, expand the tree item of the discovery site. It seems that if you don't do it, it doesn't load the content of the discovery site.
- On the right, click **Select required**. This will select all the components that are required to install the Android plugin (wst, emf, etc...).
- Click **Next**, accept the agreement, click **Install All**, and restart Eclipse.

ADB reports "no device" when an emulator is running

Try restarting adb by stopping it (`adb kill-server`) then any other adb command to restart it.

My new application/activity isn't showing up in the applications list

- You often must restart your device or emulator before a new activity shows up in the applications list. This is particularly true when it is a completely new application with a new AndroidManifest.xml file.
- If this is for a new activity in an existing AndroidManifest.xml file, did you include an `<activity>` tag for your app (or a `<service>` tag for a service, or a `<receiver>` tag for a receiver, etc.)?
- Make sure that your AndroidManifest.xml file is valid. Errors in attribute values, such as the *value* attribute in `<action value="<something>">` will often not be caught by compilers, but will prevent your application from being displayed because the intent filter will not be matched. Extra spaces or other characters can often sneak into these strings.

Did you send your .apk file to the device (adb install)?

- Run logcat on your device (`adb logcat`) and then install your .apk file. Check the logcat output to see whether the application is being installed and recognized properly. Here's sample output from a successful installation:

```
I/FileObserver( 414): *** onEvent wfd: 3 mask: 8 path: MyRSSReader.apk
D/PackageManager( 414): Scanning package: /data/app/MyRSSReader.apk
D/PackageManager( 414): Adding package com.example.codelab.rssexample
D/PackageManager( 414): Registered content provider: my_rss_item, className =
com.example.codelab.rssexample.RssContentProvider, isSyncable = false
D/PackageManager( 414): Providers: com.example.codelab.rssexample.RssContentProvider
D/PackageManager( 414): Activities: com.example.codelab.rssexample.MyRssReader
com.example.codelab.rssexample.MyRssReader2
```

- If logcat shows that the package manager is having problems loading the manifest file, force your manifest to be recompiled by adding a space in the file and compiling it.

I updated my app, but the updates don't seem to be showing up on the device

Did you remember to send your .apk file to the device (adb install)?

I'm getting a "Binary XML file line #2: You must supply a layout_wilih attribute" error when I start an application (but I declare a layout_wilih attribute *right there!!!*)

- Make sure that the SDK you are building with is the same version as the Android OS that you are running on.
- Make sure that you're calling setContentView() early in your onCreate() method. Calling other methods, such as setListAdapter() before calling setContentView() can sometimes create odd errors when Android tries to access screen elements that haven't been set before.

My request to (*make a call, catch an incoming SMS, receive a notification, send an intent to an Android application*) is being ignored

You might not have permission (or might not have requested permission) to call this activity or receive this intent. Many standard Android activities, such as making a call, have a permission assigned to it to prevent arbitrary applications from sending or receiving requests. See Security and Permissions for more information on permissions, and Manifest.permission for a list of permissions enforced by Android.

Help! My project won't build in Eclipse

If your project doesn't build, you may notice symptoms such as new resources added in the `res/` sub-folders not showing up in the R class, the emulator not being started, not being able to run the application, or even seeming to run an old version of the application.

To troubleshoot these types of problems, first try:

1. Switch to the DDMS view in Eclipse (if you don't already have it open):
   a. From the menu select `Window > Open Perspective > Other`
   b. Select DDMS from the list and hit OK
2. In the Devices panel (top right panel by default), click on the down triangle to bring up the panel menu
3. Select `Reset ADB` from the menu, and then try running the application again

If the above still doesn't work, you can try these steps:

1. Check the console and problems tabs at the bottom of the Eclipse UI
2. If there are problems listed in either place, they should give you a clue what is wrong
3. If you aren't sure if the problems are fresh or stale, clear the console with a right click > Clear, then clean the project
4. To clean the project (a good idea with any kind of build error), select Project > Clean from the eclipse main menu, then select the project you are working on (or clean all)

Eclipse isn't talking to the emulator

When communication doesn't seem to be happening between Eclipse and the emulator, symptoms can include: nothing happening when you press run, the emulator hanging waiting for a debugger to connect, or errors that Eclipse reports about not being able to find the emulator or shell. By far the most common symptom is that when you press run, the emulator starts (or is already running), but the application doesn't start.

You may find any of these steps will fix the problem and with practice you probably can figure out which one you need to do for your particular issue, but to start with, the safest option is to run through all of them in order:

1. Quit the emulator if it is running
2. Check that any emulator processes are killed (sometimes they can hang, use ps on unix or mac, or task manager in the process view on windows).
3. Quit Eclipse
4. From the command line, type:

```
adb kill-server
```

5. Start Eclipse and try again

When I go to preferences in Eclipse and select "Android", I get the following error message: Unsupported major.minor version 49.0.

This error is displayed if you are using an older version of the JDK. Please make sure you are using JDK version 5 or 6.

I can't install ApiDemos apps in my IDE because of a signing error

The Android system requires that all applications be signed, as described in Signing Your Applications. The ApiDemos applications included with the SDK are preinstalled on the emulator and for that reason have been compiled and signed with a private key.
If you want to modify or run one of the ApiDemos apps from Eclipse/ADT or other IDE, you can do so so only after you uninstall the *preinstalled* version of the app from the emulator. If you try to run an ApiDemos apps from your IDE without removing the preinstalled version first, you will get errors similar to:

```
[2008-08-13 15:14:15 - ApiDemos] Re-installation failed due to different application signatures.
[2008-08-13 15:14:15 - ApiDemos] You must perform a full uninstall of the application. WARNING: ...This
will remove the application data!
[2008-08-13 15:14:15 - ApiDemos] Please execute 'adb uninstall com.android.samples' in a shell.
```

The error occurs because, in this case, you are attempting to install another copy of ApiDemos onto the emulator, a copy that is signed with a different certificate (the Android IDE tools will have signed the app with a debug certificate, where the existing version was already signed with a private certificate). The system does not allow this type of reinstallation.

To resolve the issue, you need to fully uninstall the preinstalled and then reinstall it using the adb tool. Here's how to do that:

1. In a terminal, change to the tools directory of the SDK.
2. If no emulator instance is running, start an emulator using using the command `emulator &`.
3. Uninstall the preinstalled app using the command `adb uninstall com.android.samples`.
4. Reinstall the app using the command `adb install <path to the ApiDemos.apk>`. If you are working in Eclipse/ADT, you can just compile and run the app in the normal way.

Note that if multiple emulator instances are running, you need to direct your uninstall/install commands to the emulator instance that you are targeting. To do that you can add the `-s <serialNumber>` to the command, for example:

```
adb -s emulator-5556 install
```

For more information about adb, see the Android Debug Bridge documentation.

I can't compile my app because the build tools generated an expired debug certificate

If your development machine uses a locale that has a non-Gregorian calendar, you may encounter problems when first trying to compile and run your application. Specifically, you may find that the Android build tools won't compile your application because the debug key is expired.

The problem occurs because the Keytool utility — included in the JDK and used by the Android build tools — fails to properly handle non-Gregorian locales and may create validity dates that are in the past. That is, it may generate a debug key that is already expired, which results in the compile error.

If you encounter this problem, follow these steps to work around it:

1. First, delete the debug keystore/key already generated by the Android build tools. Specifically, delete the `debug.keystore` file. On Linux/Mac OSX, the file is stored in `~/.android`. On Windows XP, the file is stored in `C:\Documents and Settings\<user>\Local Settings\Application Data\Android`. On Windows Vista, the file is stored in `C:\Users\<user>\AppData\Local\Android`
2. Next, you can either
   - Temporarily change your development machine's locale (date and time) to one that uses a Gregorian calendar, for example, United States. Once the locale is changed, use the Android build tools to compile and install your app. The build tools will regenerate a new keystore and debug key with valid dates. Once the new debug key is generated, you can reset your development machine to the original locale.
   - Alternatively, if you do not want to change your machine's locale settings, you can generate the keystore/key on any machine using the Gregorian calendar, then copy the `debug.keystore` file from that computer to the proper location on your development machine.

This problem has been verified on Windows and may apply to other platforms.

For general information about signing Android applications, see Signing Your Applications.

I can't run a JUnit test class in Eclipse/ADT

If you are developing on Eclipse/ADT, you can add JUnit test classes to your application. However, you may get an error when trying to run such a class as a JUnit test:

```
Error occurred during initialization of VM
java/lang/NoClassDefFoundError: java/lang/ref/FinalReference
```

This error occurs because android.jar does not include complete Junit.* class implementations, but includes stub classes only.

To add a JUnit class, you have to set up a JUnit configuration:.

1. In the Package Explorer view, select your project.
2. Open the launch configuration manager.
   - In Eclipse 3.3 (Europa), select **Run** > **Open Run Dialog...** or **Run** > **Open Debug Dialog...** .
   - In Eclipse 3.4 (Ganymede), select **Run** > **Run Configurations...** or **Run** > **Debug Configurations...** .
3. In the configuration manager, right-click the "JUnit" configuration type and select **New**
4. In the new configuration's **Test** tab, specify the project and test class, as well as any options for running the test.
5. In the new configuration's **Classpath** tab, find "Android Library" under Bootstrap Entries and remove it.
6. Still in the **Classpath** tab, select Bootstrap Entries and click the Advanced button.
   a. Choose Add Library and click OK.
   b. Select JRE System Library and click Next.
   c. Select Workspace Default JRE and click Finish.
7. Select Bootstrap Entries again and click Advanced.
   a. Choose Add Library and click OK.
   b. Select JUnit 3 and click Finish.

When configured in this way, your JUnit test class should now run properly.

 Build 110632-110632 - 22 Sep 2008 13:34

Android

Open Source Licensing

Where can I find the open source components of Android?

You can find the kernel at `http://git.android.com` and the other mirrored GPL and LGPL'd components at `http://code.google.com/p/android/downloads/list`.

Notices for other licenses can be found within the SDK.

When will we see more code released under open source licenses?

Over time, more of the code that makes up Android will be released, but at this point, we have been concentrating on shipping an SDK that helps application developers get started. In short: Stay tuned.

Why are you releasing the code under the Apache License instead of GPLv2?

One of the best explanations for the reasoning behind releasing code under Apache2 can be found in a ArsTechnica article by Ryan Paul.

Build 110632-110632 - 22 Sep 2008 13:34

Android

Android Application Framework

Do all the Activities and Services of an application run in a single process?

All Activities and Services in an application run in a single process by default. The android:process attribute can be used to explicitly place a component (Activity/Service) in another process.

Do all Activities run in the main thread of an application process?

By default, all of the application code in a single process runs in the main UI thread. This is the same thread that also handles UI events. The only exception is the code that handles IPC calls coming in from other processes. The system maintains a separate pool of transaction threads in each process to dispatch all incoming IPC calls. The developer should create separate threads for any long-running code, to avoid blocking the main UI thread.

How do I pass data between Activities/Services within a single application?

It depends on the type of data that you want to share:

Primitive Data Types

To share primitive data between Activities/Services in an application, use Intent.putExtras(). For passing primitive data that needs to persist use the Application Preferences.

Non-Persistent Objects

For sharing complex non-persistent user-defined objects for short duration, the following approaches are recommended:

The android.app.Application class

The android.app.Application is a base class for those who need to maintain global application state. It can be accessed via getApplication() from any Activity or Service. It has a couple of life-cycle methods and will be instantiated by Android automatically if your register it in AndroidManifest.xml.

A public static field/method

An alternate way to make data accessible across Activities/Services is to use *public static* fields and/or methods. You can access these static fields from any other class in your application. To share an object, the activity which creates your object sets a static field to point to this object and any other activity that wants to use this object just accesses this static field.

A HashMap of WeakReferences to Objects

You can also use a HashMap of WeakReferences to Objects with Long keys. When an activity wants to pass an object to

another activity, it simply puts the object in the map and sends the key (which is a unique Long based on a counter or time stamp) to the recipient activity via intent extras. The recipient activity retrieves the object using this key.

A Singleton class

There are advantages to using a static Singleton, such as you can refer to them without casting getApplication() to an application-specific class, or going to the trouble of hanging an interface on all your Application subclasses so that your various modules can refer to that interface instead.

But, the life cycle of a static is not well under your control; so to abide by the life-cycle model, the application class should initiate and tear down these static objects in the onCreate() and onTerminate() methods of the Application Class

Persistent Objects

Even while an application appears to continue running, the system may choose to kill its process and restart it later. If you have data that you need to persist from one activity invocation to the next, you need to represent that data as state that gets saved by an activity when it is informed that it might go away.

For sharing complex persistent user-defined objects, the following approaches are recommended:

- Application Preferences
- Files
- contentProviders
- SQLite DB

If the shared data needs to be retained across points where the application process can be killed, then place that data in persistent storage like Application Preferences, SQLite DB, Files or ContentProviders. Please refer to the Storing, Retrieving and Exposing Data for further details on how to use these components.

How can I check if an Activity is already running before starting it?

The general mechanism to start a new activity if its not running— or to bring the activity stack to the front if is already running in the background— is the to use the NEW_TASK_LAUNCH flag in the startActivity() call.

If an Activity starts a remote service, is there any way for the Service to pass a message back to the Activity?

The remote service can define a callback interface and register it with the clients to callback into the clients. The RemoteCallbackList provides methods to register and unregister clients with the service, and send and receive messages.

The sample code for remote service callbacks is given in ApiDemos/RemoteService

How to avoid getting the Application not responding dialog?

Please check the Application Responsiveness section to design your application for better responsiveness:

How does an application know if a package is added or removed?

Whenever a package is added, an intent with PACKAGE_ADDED action is broadcast by the system. Similarly when a package is removed, an intent with PACKAGE_REMOVED action is broadcast. To receive these intents, you should write something like this:

```
<receiver android:name ="com.android.samples.app.PackageReceiver">
     <intent-filter>
      <action android:name="android.intent.action.PACKAGE_ADDED"/>
       <action android:name="android.intent.action.PACKAGE_REMOVED"/>
```

```
            <data android:scheme="package" />
        </intent-filter>
    </receiver>
```

Here PackageReceiver is a BroadcastReceiver class.Its onReceive() method is invoked, every time an application package is installed or removed.

 Build 110632-110632 - 22 Sep 2008 13:34

Android

Security

Is Android secure?

The security and privacy of our users' data is of primary importance to the Android Open Source Project. We are dedicated to building and maintaining one of the most secure mobile platforms available while still fulfilling our goal of opening the mobile device space to innovation and competition.

The Android Platform provides a rich security model that allows developers to request the capabilities, or access, needed by their application and to define new capabilities that other applications can request. The Android user can choose to grant or deny an application's request for certain capabilities on the handset.

We have made great efforts to secure the Android platform, but it is inevitable that security bugs will be found in any system of this complexity. Therefore, the Android team works hard to find new bugs internally and responds quickly and professionally to vulnerability reports from external researchers.

I think I found a security flaw. How do I report it?

You can reach the Android security team at security@android.com. If you like, you can protect your message using our PGP key.

We appreciate researchers practicing responsible disclosure by emailing us with a detailed summary of the issue and keeping the issue confidential while users are at risk. In return, we will make sure to keep the researcher informed of our progress in issuing a fix and will properly credit the reporter(s) when we announce the patch. We will always move swiftly to mitigate or fix an externally-reported flaw and will publicly announce the fix once patches are available to users.

How can I stay informed of Android security announcements?

An important part of sustainably securing a platform, such as, Android is keeping the user and security community informed of bugs and fixes. We will publicly announce security bugs when the fixes are available via postings to the android-security-announce group on Google Groups. You can subscribe to this group as you would a mailing list and view the archives here.

For more general discussion of Android platform security, or how to use security features in your Android application, please subscribe to android-security-discuss.

How do I securely use my Android phone?

As an open platform, Android allows users to load software from any developer onto a device. As with a home PC, the user must be aware of who is providing the software they are downloading and must decide whether they want to grant the application the capabilities it requests. This decision can be informed by the user's judgment of the software developer's trustworthiness, and where the software came from.

Despite the security protections in Android, it is important for users to only download and install software from developers they

trust. More details on how Android users can make smart security decisions will be released when consumer devices become available.

I think I found malicious software being distributed for Android. How can I help?

Like any other open platform, it will be possible for unethical developers to create malicious software, known as malware, for Android. If you think somebody is trying to spread malware, please let us know at security@android.com. Please include as much detail about the application as possible, with the location it is being distributed from and why you suspect it of being malicious software.

The term *malicious software* is subjective, and we cannot make an exhaustive definition. Some examples of what the Android Security Team believes to be malicious software is any application that:

- drains the device's battery very quickly;
- shows the user unsolicited messages (especially messages urging the user to buy something);
- resists (or attempts to resist) the user's effort to uninstall it;
- attempts to automatically spread itself to other devices;
- hides its files and/or processes;
- discloses the user's private information to a third party, without the user's knowledge and consent;
- destroys the user's data (or the device itself) without the user's knowledge and consent;
- impersonates the user (such as by sending email or buying things from a web store) without the user's knowledge and consent; or
- otherwise degrades the user's experience with the device.

How will Android-powered devices receive security fixes?

The manufacturer of each device is responsible for distributing software upgrades for it, including security fixes. Many devices will update themselves automatically with software downloaded "over the air", while some devices require the user to upgrade them manually.

When Android-powered devices are publicly available, this FAQ will provide links how Open Handset Alliance members release updates.

Can I get a fix directly from the Android Platform Project?

Android is a mobile platform that will be released as open source and available for free use by anybody. This means that there will be many Android-based products available to consumers, and most of them will be created without the knowledge or participation of the Android Open Source Project. Like the maintainers of other open source projects, we cannot build and release patches for the entire ecosystem of products using Android. Instead, we will work diligently to find and fix flaws as quickly as possible and to distribute those fixes to the manufacturers of the products.

In addition, We will add security fixes to the open source distribution of Android and publicly announce the changes on android-security-announce.

If you are making an Android-powered device and would like to know how you can properly support your customers by keeping abreast of software updates, please contact us at info@openhandsetalliance.com.