Get Started Developing for Android with Eclipse

URL: http://www.smashingmagazine.com/2010/10/25/get-started-developing-for-android-with-eclipse/

By Chris Blunt | October 25th, 2010 | Coding | 48 Comments | Publishing Policy |

There's a lot to get excited about in mobile application development today. With increasingly sophisticated hardware, tablet PCs and a variety of software platforms (Symbian OS, iOS, WebOS, Windows Phone 7…), the landscape for mobile developers is full of opportunities — and a little complex as well.

So much choice can be overwhelming when you just want to **get started building mobile applications**. Which platform should you choose? What programming language should you learn? What kit do you need for your planned project? In this tutorial, you'll learn how to start writing applications for Android *(http://android.com/)*, the open-source mobile operating system popularized by Google.

Why Develop for Android?

Android is an open-source platform based on the Linux kernel, and is installed on thousands of devices *(http://mashable.com/2010/09/16/android-comscore-july-2010/)* from a wide range of manufacturers. Android exposes your application to all sorts of hardware that you'll find in modern mobile devices — digital compasses, video cameras, GPS, orientation sensors, and more.

Android's free development tools make it possible for you to start writing software at little or no cost. When you're ready to show off your application to the world, you can publish it to Google's Android Market. **Publishing to Android Market** incurs a one-off registration fee (US $25 at the time of writing) and, unlike Apple's App Store which famously reviews each submission, makes your application available for customers to download and buy after a quick review process — unless the application is blatantly illegal.

Here are a few other advantages Android offers you as a developer:

- The Android SDK is available for Windows, Mac and Linux, so you don't need to pay for new hardware to start writing applications.
- An SDK built on Java. If you're familiar with the Java programming language, you're already halfway there.
- By distributing your application on Android Market, it's available to hundreds of thousands *(http://www.wired.com/gadgetlab/2010/08/google-200000-android-phones/)* of users instantly. You're not just limited to one store, because there are alternatives, too. For instance, you can release your application on your own blog. Amazon have recently been rumoured *(http://www.slashgear.com/amazon-android-app-store-tcs-leak-29104993/)* to be preparing their own Android app store also.
- As well as the technical SDK documentation *(http://developer.android.com/sdk/index.html)*, new resources are being published for Android developers as the platform gains popularity among both users and developers.

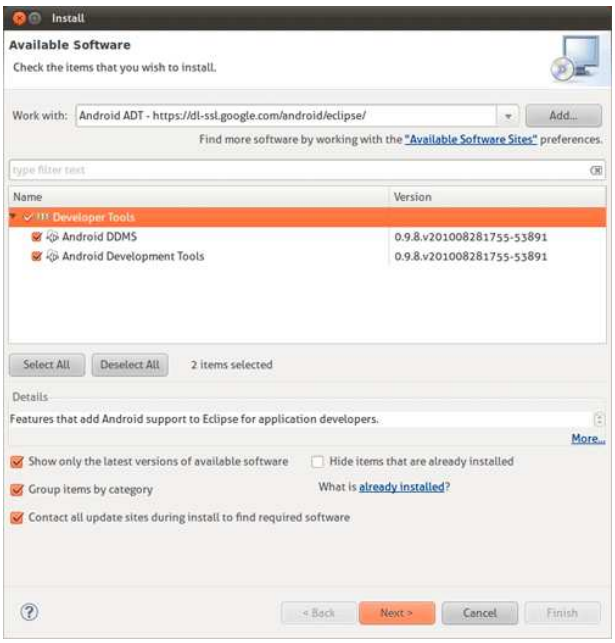
Enough with the talk — let's get started developing for Android!

Installing Eclipse and the Android SDK

The recommended environment for **developing Android applications** is Eclipse with the Android Development Toolkit (ADT) plugin installed. I'll summarize the process here. If you need more detail, Google's own developer pages *(http://developer.android.com/sdk/)* do a good job of explaining the installation and configuration process.

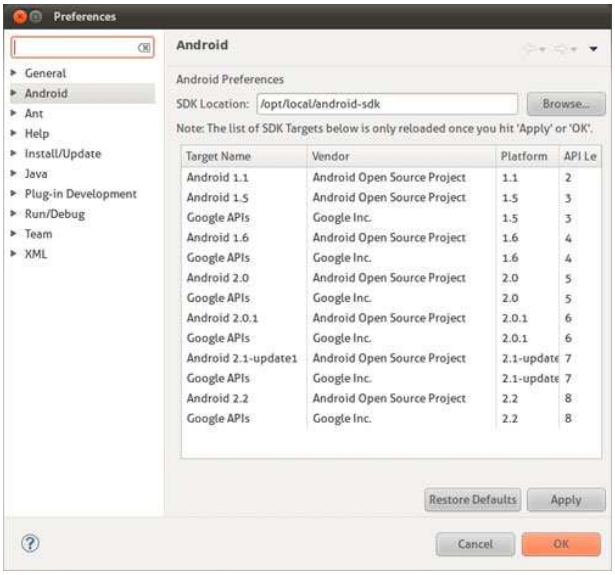
- Download the Android SDK *(http://developer.android.com/)* for your platform (Windows, Mac OS X, or Linux).
- Extract the downloaded file to somewhere memorable on your hard drive (on Linux, I use `/opt/local/`).

- Click *Add* in the Available Software window.
- Enter `Android Development Tools` in the *Name* field, and `https://dl-ssl.google.com/android/eclipse/` in the *Location* field.
- Click *OK* and check *Developer Tools* in the list of available software. This will install the Android Development Tools and DDMS, Android's debugging tool.



(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/eclipse_install_adt.jpg)
Large image (http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/eclipse_install_adt.jpg)

- Click *Next* and *Finish* to install the plugin. You'll need to restart Eclipse once everything is installed.
- When Eclipse restarts, choose *Window->Preferences* and you should see *Android* listed in the categories.
- You now need to tell Eclipse where you've installed the Android SDK. Click *Android* and then *Browse* to select the location where you extracted the SDK files. For example, `/opt/local/android-sdk`.



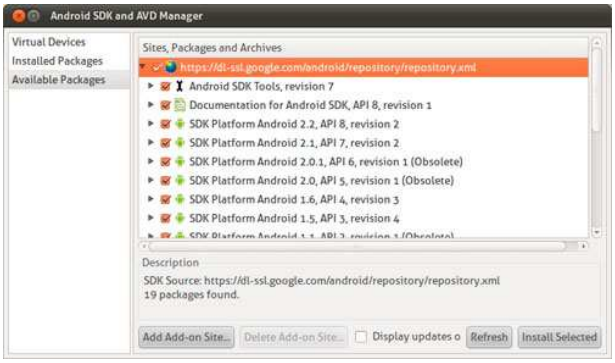
(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/eclipse_android_preferences.jpg)
Large view (http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/eclipse_android_preferences.jpg)

- Click *OK* to have Eclipse save the location of your SDK.

devices for which you want to develop apps. Each platform has a different version of the Android SDK that may be installed on users' devices. For versions of Android 1.5 and above, there are two platforms available: *Android Open Source Project* and *Google*.

The *Android Open Source Project* platforms are open source, but do not include Google's proprietary extensions such as *Google Maps*. If you choose not to use the Google APIs, Google's mapping functionality won't be available to your application. Unless you have a specific reason not to, I'd recommended you to target one of the Google platforms, as this will allow you to take advantage of Google's proprietary extensions.

- Choose *Window->Android SDK and AVD Manager*.
- Click *Available Packages* in the left column and check the repository to show a list of the available Android platforms.
- You can choose which platforms to download from the list, or leave everything checked to download all the available platforms. When you're done, click *Install Selected* and follow the installation instructions.



(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/sdk_manager_platforms.jpg)
Large image (http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/sdk_manager_platforms-550-e1287474433673.jpg)

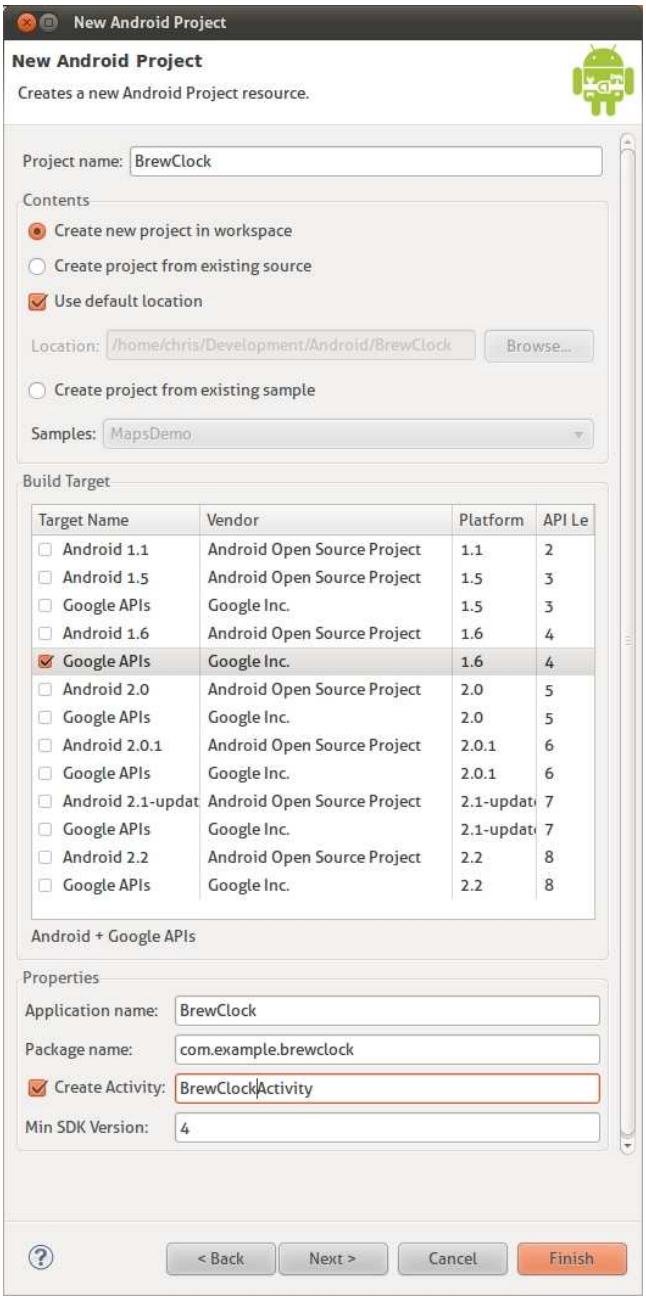
Once everything has been successfully downloaded, you're ready to start developing for Android.

Creating a New Android Project

Eclipse's New Project Wizard can create a new Android application for you, generating files and code that are ready to run right out of the box. It's a quick way to see something working, and a good starting point from which to develop your own applications:

- Choose *File->New->Project…*
- Choose *Android Project*
- In the *New Project* dialog, enter the following settings:

```
Project Name: BrewClock
Build Target: Google Inc. 1.6 (Api Level 4)
Application Name: BrewClock
Package Name: com.example.brewclock
Create Activity: BrewClockActivity
Min SDK Version: 4
```



(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/eclipse_new_project_settings.jpg)

After clicking *Finish*, Eclipse will create a new Android project that's ready to run. Notice you told Eclipse to generate an Activity called `BrewClockActivity`? This is the code that Android actually uses to run your application. The generated code will display a simple 'Hello World' style message when the application runs.

Packages

The package name is an identifier for your application. When the time comes and you are willing to publish on Android Market, it's exactly this identifier that will be used to track your application for updates, so it's important to **make sure it's unique**. Although we're using the `com.example.brewclock` namespace here, for a real application it's best to choose something like `com.yourcompanyname.yourapplication`.

SDK Versions

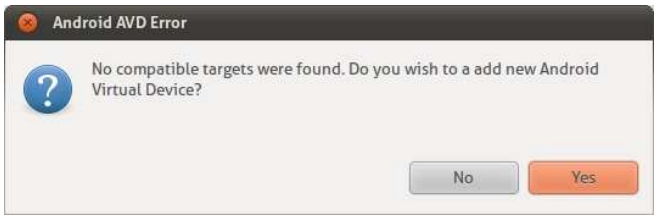
The `Min SDK Version` is the earliest version of Android on which your application will run. With each

Running Your Application

Now let's try running the application in Eclipse. As this is the first run, Eclipse will ask what type of project you are working on:

- Choose *Run->Run* or press *Ctrl+F11*.
- Choose *Android Application* and click *OK*.

Eclipse will now try to run the application on an Android device. At the moment, though, you don't have any Android devices running, so the run will fail and you'll be asked to create a new *Android Virtual Device* (AVD).



Android Virtual Devices

An Android Virtual Device (AVD) is an emulator that simulates a real-world Android device, such as a mobile phone or Tablet PC. You can use AVDs to test how your application performs on a wide variety of Android devices, without having to buy every gadget on the market.

You can create as many AVDs as you like, each set up with different versions of the Android Platform. For each AVD you create, you can configure various hardware properties such as whether it has a physical keyboard, GPS support, the camera resolution, and so on.

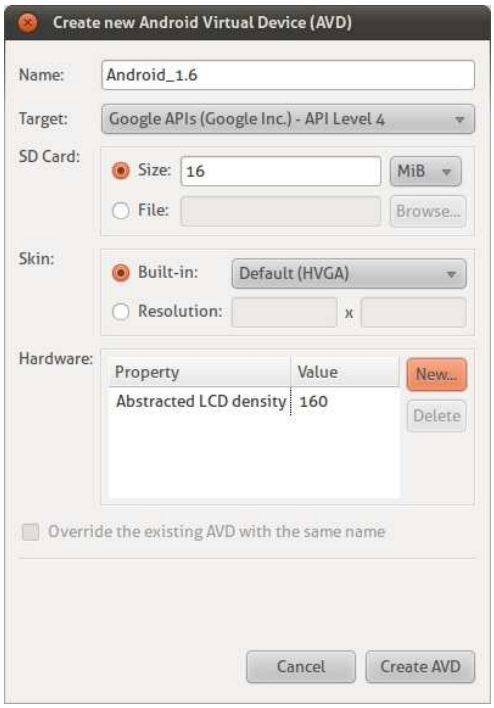
Before you can run your application, you need to create your first AVD running the target SDK platform (Google APIs 1.6).

Let's do that now:

- If you haven't tried to run your application yet, click *Run* now (or hit *Ctrl+F11*)
- When the target device warning pops up, click *Yes* to create a new AVD.
- Click *New* in the *Android SDK and AVD Manager* dialog that appears.
- Enter the following settings for the AVD:

```
1 Name: Android_1.6
2 Target: Google APIs (Google Inc.) - API Level 4
3 SD Card Size: 16 MiB
4 Skin Built In: Default (HVGA)
```

- Click *Create AVD* to have Android build your new AVD.
- Close the *Android SDK and AVD Manager* dialog.



(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/sdk_manager_new_avd.jpg)

Running the Code

Try running your application again (*Ctrl+F11*). Eclipse will now build your project and launch the new AVD. Remember, the AVD emulates a complete Android system, so you'll even need to sit through the slow boot process just like a real device. For this reason, once the AVD is up and running, it's best not to close it down until you've finished developing for the day.

When the emulator has booted, Eclipse automatically installs and runs your application:

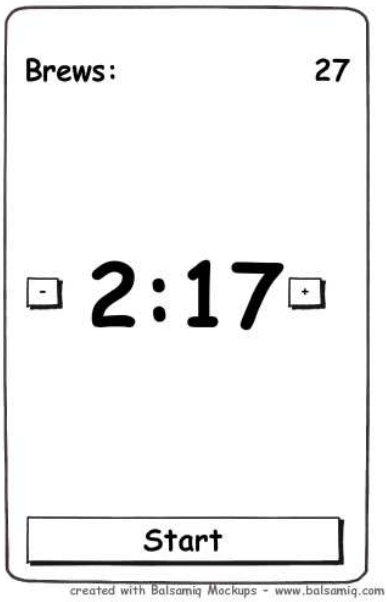
(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/app_running.jpg)
Large image (http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/app_running.jpg)

Building Your First Android Application

Testing generated code is all well and good, but you want to start building a real application. For this, we'll step through a simple design process and build an application that you can deploy to your Android device.

Most developers (myself included) like a **constant supply of good tea or coffee**. In the next section of this article you'll build a simple tea counter application to track how many cups of tea (*brews*) the user has drunk, and let them set a timer for brewing each cup.

One of the first steps to building any Android application is to design and build the user interface. Here's a quick sketch of how the application's interface will look:



(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/design_sketch.jpg)
Large image (http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/design_sketch.jpg)

The user will be able to set a brew time in minutes using the + and - buttons. When they click *Start*, a countdown will start for the specified number of minutes. Unless the user cancels the brew by tapping the button again, the brew count will be increased when the countdown timer reaches 0.

Building the Interface

Android user interfaces, or *layouts*, which are described in XML documents, can be found in the `res/layouts` folder. The template code that Eclipse generated already has a simple layout declared in `res/layouts/main.xml` which you may have seen previously while the application was running on the emulator.

Eclipse has a graphical layout designer that lets you build the interface by 'dragging' and 'dropping' controls around the screen. However, I often find it easier to write the interface in XML and use the graphical layout to preview the results.

Let's do this now by changing `main.xml` to match the design sketch above:

- Open `res/layouts/main.xml` in Eclipse by double-clicking it in the *Package Explorer*.
- Click the `main.xml` tab along the bottom of the screen to switch to XML view.

Now change the content of `main.xml` to:

```
01 # /res/layouts/main.xml
02 <?xml version="1.0" encoding="utf-8"?>
03 <LinearLayout
04   xmlns:android="http://schemas.android.com/apk/res/android
   (http://schemas.android.com/apk/res/android)"
05   android:orientation="vertical"
06   android:layout_width="fill_parent"
07   android:layout_height="fill_parent">
08   <LinearLayout
09     android:orientation="horizontal"
10     android:layout_width="fill_parent"
11     android:layout_height="wrap_content"
12     android:padding="10dip">
13     <TextView
14       android:layout_width="wrap_content"
```

```
      android:gravity="right"
      android:textSize="20dip"
      android:id="@+id/brew_count_label" />
  </LinearLayout>
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:gravity="center"
    android:padding="10dip">
    <Button
      android:id="@+id/brew_time_down"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="-"
      android:textSize="40dip" />
    <TextView
      android:id="@+id/brew_time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="0:00"
      android:textSize="40dip"
      android:padding="10dip" />
    <Button
      android:id="@+id/brew_time_up"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="+"
      android:textSize="40dip" />
  </LinearLayout>
  <Button
    android:id="@+id/brew_start"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:text="Start" />
</LinearLayout>
```

As you can see, Android's XML layout files are verbose, but allow you to control virtually every aspect of elements on the screen.

One of the most important interface elements in Android are `Layout` containers, such as the `LinearLayout` used in this example. These elements are invisible to the user but act as layout containers for other elements such as `Buttons` and `TextViews`.

There are several types of layout views, each of which is used to build different types of layout. As well as the `LinearLayout` and `AbsoluteLayout`, the `TableLayout` allows the use of complex grid-based interfaces. You can find out more about Layouts in the Common Layout Objects *(http://developer.android.com/guide/topics/ui/layout-objects.html)* section of the API documents.

Linking Your Layout With Code

After saving your layout, try running your application in the emulator again by pressing *Ctrl+F11*, or clicking the *Run* icon in Eclipse. Now instead of the 'Hello World' message you saw earlier, you'll see Android now displays your application's new interface.

If you click any of the buttons, they'll highlight as expected, but don't do anything yet. Let's remedy that by writing some code behind the interface layout:

```
# /src/com/example/brewclock/BrewClockActivity.java
...
import android.widget.Button;
import android.widget.TextView;

public class BrewClockActivity extends Activity {
  /** Properties **/
  protected Button brewAddTime;
  protected Button brewDecreaseTime;
  protected Button startBrew;
  protected TextView brewCountLabel;
  protected TextView brewTimeLabel;
```

The Resource Object

In Android, `R` is a special object that is automatically generated to allow access to your project's resources (layouts, strings, menus, icons…) from within the code. Each resource is given an `id`. In the layout file above, these are the `@+id` XML attributes. We'll use those attributes to connect the `Buttons` and `TextViews` in our layout to the code:

```
# /src/com/example/brewclock/BrewClockActivity.java
...
public class BrewClockActivity extends Activity {
  ...
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Connect interface elements to properties
    brewAddTime = (Button) findViewById(R.id.brew_time_up);
    brewDecreaseTime = (Button) findViewById(R.id.brew_time_down);
    startBrew = (Button) findViewById(R.id.brew_start);
    brewCountLabel = (TextView) findViewById(R.id.brew_count_label);
    brewTimeLabel = (TextView) findViewById(R.id.brew_time);
  }
}
```

Listening For Events

In order to detect when the user taps one of our buttons, we need to implement a listener. You may be familiar with listeners or *callbacks* from other event-driven platforms, such as Javascript/jQuery events or Rails' callbacks.

Android provides a similar mechanism by providing `Listener` interfaces, such as `OnClickListener`, that define methods to be triggered when an event occurs. Implementing the `OnClickListener` interface will notify your application when the user taps the screen, and on which button they tapped. You also need to tell each button about the `ClickListener` so that it knows which listener to notify:

```
# /src/com/example/brewclock/BrewClockActivity.java
...
// Be sure not to import
// `android.content.dialoginterface.OnClickListener`.
import android.view.View.OnClickListener;

public class BrewClockActivity extends Activity
  implements OnClickListener {
  ...
  public void onCreate(Bundle savedInstanceState) {
    ...
    // Setup ClickListeners
    brewAddTime.setOnClickListener(this);
    brewDecreaseTime.setOnClickListener(this);
    startBrew.setOnClickListener(this);
  }
  ...
  public void onClick(View v) {
    // TODO: Add code to handle button taps
  }
}
```

Next we'll add code that handles each of our button presses. We'll also add four new properties to the Activity that will let the user set and track the brewing time, how many brews have been made, and whether the timer is currently running.

```
# /src/com/example/brewclock/BrewClockActivity.java
...
public class BrewClockActivity extends Activity
  implements OnClickListener {
  ...
  protected int brewTime = 3;
  protected CountDownTimer brewCountDownTimer;
  protected int brewCount = 0;
  protected boolean isBrewing = false;
```

```
      if(isBrewing)
        stopBrew();
      else
        startBrew();
    }
  }
}
```

Notice we're using the `CountDownTimer` class provided by Android. This lets you easily create and start a simple countdown, and be notified at regular intervals whilst the countdown is running. You'll use this in the `startBrew` method below.

The following methods are all model logic that handles setting the brew time, starting and stopping the brew and maintaining a count of brews made. We'll also initialize the `brewTime` and `brewCount` properties in `onCreate`.

It would be good practice to move this code to a separate model class, but for simplicity we'll add the code to our `BrewClockActivity`:

```
# /src/com/example/brewclock/BrewClockActivity.java
...
public class BrewClockActivity extends Activity
  implements OnClickListener {
  ...
  public void onCreate(Bundle savedInstanceState) {
    ...
    // Set the initial brew values
    setBrewCount(0);
    setBrewTime(3);
  }

  /**
   * Set an absolute value for the number of minutes to brew.
   * Has no effect if a brew is currently running.
   * @param minutes The number of minutes to brew.
   */
  public void setBrewTime(int minutes) {
    if(isBrewing)
      return;

    brewTime = minutes;

    if(brewTime < 1)
      brewTime = 1;

    brewTimeLabel.setText(String.valueOf(brewTime) + "m");
  }

  /**
   * Set the number of brews that have been made, and update
   * the interface.
   * @param count The new number of brews
   */
  public void setBrewCount(int count) {
    brewCount = count;
    brewCountLabel.setText(String.valueOf(brewCount));
  }

  /**
   * Start the brew timer
   */
  public void startBrew() {
    // Create a new CountDownTimer to track the brew time
    brewCountDownTimer = new CountDownTimer(brewTime * 60 * 1000, 1000) {
      @Override
      public void onTick(long millisUntilFinished) {
        brewTimeLabel.setText(String.valueOf(millisUntilFinished / 1000) + "s");
      }

      @Override
      public void onFinish() {
        isBrewing = false;
        setBrewCount(brewCount + 1);

        brewTimeLabel.setText("Brew Up!");
        startBrew.setText("Start");
```

```
65
66    /**
67     * Stop the brew timer
68     */
69    public void stopBrew() {
70      if(brewCountDownTimer != null)
71        brewCountDownTimer.cancel();
72
73      isBrewing = false;
74      startBrew.setText("Start");
75    }
76    ...
77  }
```

The only parts of this code specific to Android are setting the display labels using the `setText` method. In `startBrew`, we create and start a `CountDownTimer` to start counting down every second until a brew is finished. Notice that we define `CountDownTimer's` listeners (`onTick` and `onFinish`) inline. `onTick` will be called every 1000 milliseconds (1 second) the timer counts down, whilst `onFinish` is called when the timer reaches zero.

Avoiding Hard-Coded Text in your Code

To keep this tutorial code simple, I've intentionally written label strings directly in the code (e.g. `"Brew Up!"`, `"Start"`, `"Stop"`). Generally, this isn't good practice, as it makes finding and changing those strings harder in large projects.

Android provides a neat way to keep your text strings separate from code with the `R` object. `R` lets you define all your application's strings in an xml file (`res/values/strings.xml`) which you can then access in code by reference. For example:

```
1  # /res/values/strings.xml
2  <string name="brew_up_label">Brew Up!</string>
3  ...
4
5  # /res/com/example/brewclock/BrewClockActivity.java
6  ...
7  brewLabel.setText(R.string.brew_up_label);
8  ...
```

Now if you wanted to change `Brew Up!` to something else, you would only need to change it once in the *strings.xml* file. Your application starts to span dozens of code files which keeps all your strings in one place and makes a lot of sense!

Trying BrewClock

With the code complete, it's time to try out the application. Hit *Run* or *Ctrl+F11* to start BrewClock in the emulator. All being well, you'll see the interface set up and ready to time your tea brewing! Try setting different brew times, and pressing *Start* to watch the countdown.

(http://media.smashingmagazine.com/cdn_smash/wp-content/uploads/2010/10/app_finished.jpg)
Large image (http://media.smashingmagazine.com/cdn_smash/wp-

(ADT) plugin. You've set up an emulator, or virtual device that can test your applications. You've also built a working Android application which has **highlighted a number of key concepts** that you'll use when developing your own Android applications.

Hopefully, this has whet your appetite for building mobile applications, and experimenting in this exciting field. Android offers a great way to start writing applications for a range of current and upcoming mobile devices. If you've built or are working on your own mobile app, be sure to let us know about it in the comments!

(ik), (vf)

Chris Blunt

Chris is a software developer working with Ruby, Rails and Android. In 2010, he founded Plymouth Software where he designs and builds applications for the web and mobile devices. As well as a fondness for travel and drinking tea, Chris writes about code, design and business on his blog at chrisblunt.com.

Homepage ***(http://chrisblunt.com/)*** **Twitter Page** ***(http://twitter.com/cblunt)***

Tags: **android**, **mobile**

48 Comments Best Comments

Martin *October 25th, 2010 5:53 am* 1
Great! Thank you!
0

Raju *October 25th, 2010 5:58 am* 2
Cool..Thanks
0

Tom Hermans *October 25th, 2010 6:02 am* 3
Great article !
Like to see *a lot* more of this !

Already followed a 1-day seminar on building Android apps, and it's not that easy for someone like me (webdesigner/developer).
Step-by-step tutorials of Smashing quality would come in handy indeed. (or another e-book maybe ?)

+9

Jorgen *October 25th, 2010 6:57 am* 4
I totally agree with Tom. I'd love to see a lot more of these type of articles!
+2

huzz *October 26th, 2010 1:22 am* 5
I agree too :)
0

eddt *October 25th, 2010 6:02 am* 6
This is a really GREAT snapshot of android development! Smashing!!!

0

Angelfire *October 25th, 2010 6:17 am* 8

Sooo much thanks, this is a great tutorial for begginers!!!!

Greetings from Colombia!!!

+1

Karl *October 25th, 2010 6:34 am* 9

Sorry for diversion, but what desktop font are you using on Ubuntu? It's really nice!

+1

Klesus *October 25th, 2010 6:48 am* 10

Sorry for ranting but I can't let this slip. "An SDK is built on Java" should probably be "THIS SDK is built on Java". Describing SDK as something that is written for Java is just… wrong, as an SDK could be for any language.

+1

Chris Blunt *October 25th, 2010 7:01 am* 11

Thanks to everyone for your comments!

@Karl The font I'm using now is "Ubuntu", the new default in 10.10 Maverick. The designers, DaltonMaag, have some other stunning fonts available.

@Klesus Thanks for pointing out the typo, I've updated the post.

+1

Fábio Santos *October 25th, 2010 7:08 am* 12

Just in time =D

I think Android is getting everyday more and more developers, and that post have become just in the right time for me =D

Maybe one day you guys start a new site, droidtuts+

TKS!

+1

Brett Kromkamp *October 25th, 2010 7:16 am* 13

Checkout the following Android tutorial:
http://www.quesucede.com/page/show/id/conway_game_of_life_android.

It walks the developer through building Conway's Game of Life – the problem space is small enough to allow the developer to only focus on Google Android while still including sufficient elements to make for both an interesting and valid learning experience.

Brett Kromkamp

0

ximo *October 25th, 2010 7:28 am* 14

Thanks for this article!
Though I think I saw "Comic Sans" somewhere…
=S

+1

Mahawk Kalkya *October 25th, 2010 11:27 am* 15

Thanks for this! How upward compatible are the different platforms? If I decide to write something for 1.6 (your example) will it run on 2.2 as well? I guess this shouldn't be a problem but I'm completely new to Android so don't shoot me :-)

0

Daniel *October 25th, 2010 10:39 am* 17

Upward compatibiliy is not a problem yet, but some things are deprecated and could not be available on future phones.

-1

ZizzelDaZuz *October 25th, 2010 10:51 pm* 18

Thanks, Daniel!

0

Laura *October 25th, 2010 7:54 am* 19

Useful stuff, thanks :) I have to ask if you used any of the articles here: http://mikeyhogarth.wordpress.com/ as a reference? They cover a lot of the same points.

-1

Sam *October 25th, 2010 7:56 am* 20

Sweet post Chris. Once devs are ready to publish and would like extra help on promoting their app(s) they should take a peek at the lightweight GamerShots SDK to include in their projects – http://gamershots.com/developer

+2

Chris Blunt *October 25th, 2010 9:00 am* 21

@ZizzelDaZuz It depends on what features of the SDK your app uses, but my experience has been that Android has good upward-compatibility.

I've run into minor problems such as changing file storage paths in 2.2, but the changes are very well-covered in the SDK docs. Your code can query the current device to discover platform and capabilities, and act appropriately.

+1

D. Troy *October 25th, 2010 9:22 am* 22

A really clear, concise and well explained tutorial, which has proven to be extremely helpful as I've been struggling getting started with Android. More from this author please!

+2

Dhruv *October 25th, 2010 9:56 am* 23

Just Awesome

+1

pioSko *October 25th, 2010 11:27 am* 24

Great article. Thank you.
I'm just starting with Android… with mobile apps… and this was perfect timing.

I have a few errors popping up in Eclipse. I think I followed your tutorial to the dot, but I'm not sure now. would it be possible for you to post the full contents of BrewClockActivity.java? This may help others, aswell :)

0

Scott *October 25th, 2010 11:34 am* 25

@pioSko: Thanks :) The source code for the tutorial is available at http://github.com/cblunt/brewclock

@Scott: Not tried PhoneGap – looks interesting though!

0

jcesar *October 25th, 2010 3:45 pm* 27

I followed your tutorial and it is very useful, thank you very much
I found a mistake, you writed
brewLabel.setText(R.string.brew_up_label);
but it should be
brewTimeLabel.setText(R.string.brew_up_label);

It is very complete, but I miss a point where you explain how to put the app image.

A tip for the web developers interested in developing android apps, take a look to phonegap, it is very useful for creating “native” iphone and android apps using html, css and javascript (and even blackberry, wm and symbian)

-1

Lester Bambico *October 25th, 2010 4:21 pm* 28

I have a feeling this article was written to somehow express a “hate” toward Jobs’ recent prescon. Great response (slap) in his face.
Keep up the good work SM.

-2

Charles *October 25th, 2010 5:16 pm* 29

That’s nice!

0

Web Designer Houston *October 25th, 2010 8:06 pm* 30

This is a really GREAT snapshot of android development! Smashing!!!

I totally agree with Tom. I’d love to see a lot more of these type of articles!
Thanks for sharing this!!!

0

rehaan *October 25th, 2010 9:56 pm* 31

great article i really want to try Android :)

+1

Chris Blunt *October 25th, 2010 10:49 pm* 32

Thanks, really appreciate all the comments! :)

@Laura Hadn’t seen that blog, but would like to add it to a list of useful resources for Android developers that I’m compiling.

0

Laura *October 26th, 2010 5:41 am* 33

Sounds good – will keep an eye out for that! :)

0

Laura *October 26th, 2010 6:06 am* 34

Oh yeah, one thing I thought might be handy to mention in your article was this

Cheers again for the article!

0

W3Planting *October 25th, 2010 11:05 pm* 35

Great article although i installed the Eclipse & Android on my PC but really cant build anything special i got inspired and get a direction on how to build a application after looking at your sample demo :)

+1

Eko S *October 25th, 2010 11:25 pm* 36

Wow, this a great post….I'm interested in becoming an Android developer.
Thanks…

0

alantakashi *October 26th, 2010 1:21 am* 37

Awesome tutorial. Thanks Chris!

0

Dirk *October 26th, 2010 2:42 am* 38

Just what i was looking for. Thanks a lot!

+1

Magnus *October 26th, 2010 3:06 am* 39

Thank you very much for this! More Android stuff @ SM!

+1

Marek *October 26th, 2010 3:39 am* 40

Thanks!
Very useful article :)
Looking fwd to more android tips

Anoyne here knows how to connect HTC Hero 2.1 with IntelliJ Idea [windows] into debug mode? Win can`t detect phone in that mode :-/

0

Tim Pelling *October 26th, 2010 4:04 am* 41

This is a really useful article, thanks

I have used Titanium by Appcelerator (http://www.appcelerator.com) to develop apps for Android and iPhone. It works well and is a good way to develop apps for multiple platforms. Maybe Smashing Mag can do an article on Titanium?

0

Day *October 26th, 2010 4:10 am* 42

Brilliant article, more like this please! :)

0

Leyton Jay *October 26th, 2010 4:12 am* 43

Perfect, I was looking for a novice intro for Eclipse on Windows. Thanks again!

+2

Ozzy *October 26th, 2010 4:14 am*

Thanks a lot for posting this tutorial, i would love to see more articles on Android development here.

0

K@reem *October 26th, 2010 11:38 am* 46

Thanks a lot its so helpful tutorial.

0

alex *October 26th, 2010 4:24 pm* 47

very good tutorials for starting with.
Thanks for sharing.:D

0

alex *October 26th, 2010 9:15 pm* 48

tx, any chance of converting this tutorial with Netbeans???

0