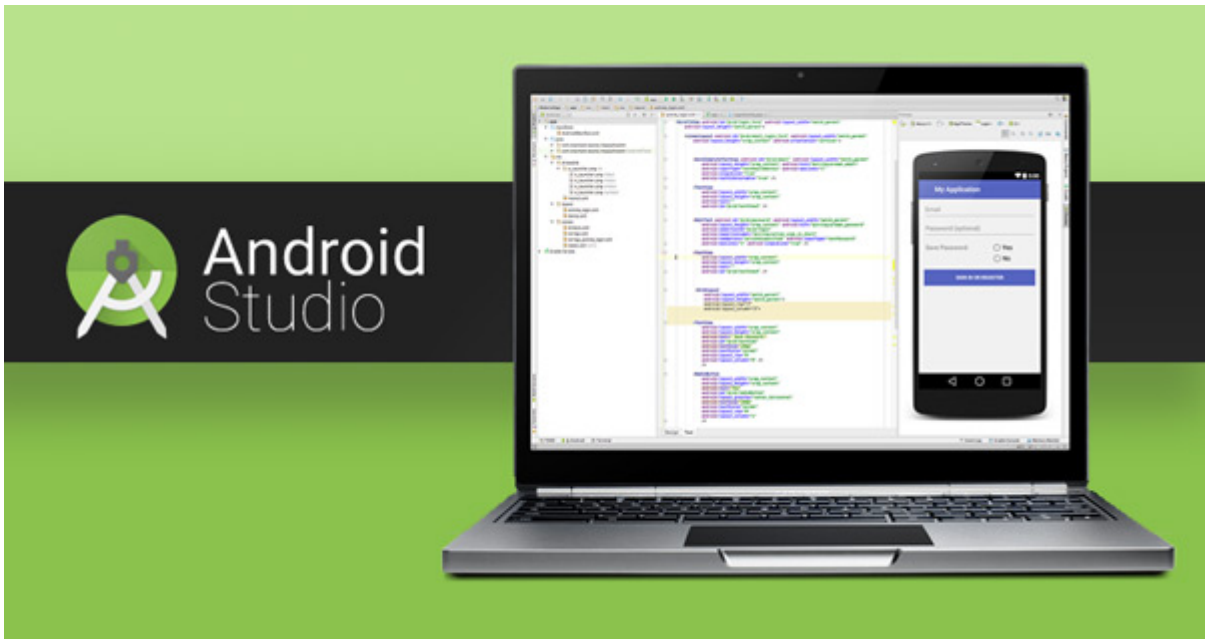
Configurando Android Studio



Introducción

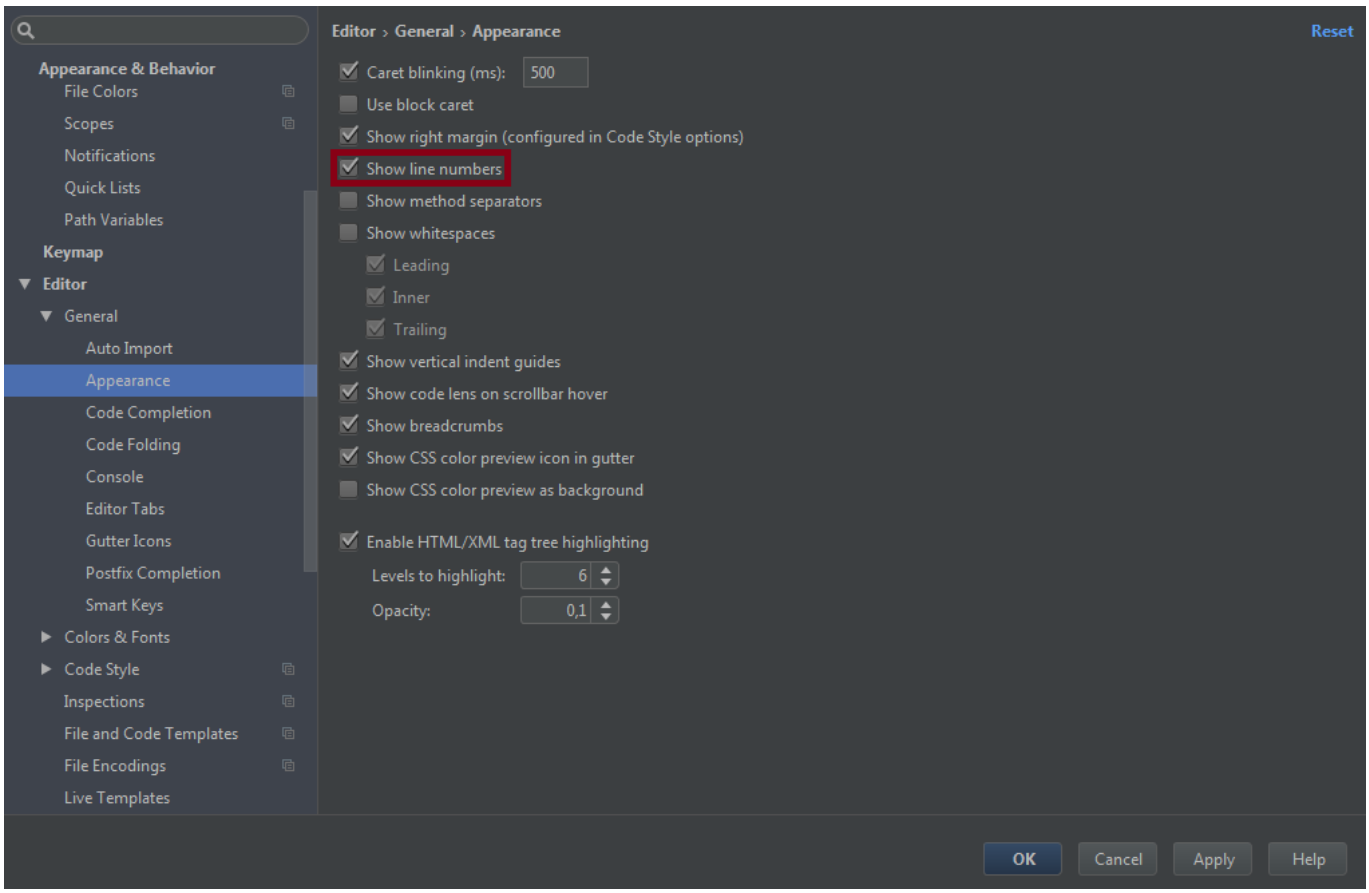
Este es el primer post de una pequeña serie sobre Android Studio. En ellos mostraremos alguna configuraciones,algún plugin, atajos de teclados y «truquillos».

- Número de linea
- Camel humps
- Imports automáticos
- Colores en el logview de Android Studio
- Plugins
    - Plugin ButterKnife Zelezny
    - Plugin Exynap
    - Markdown Navigator

Números de linea

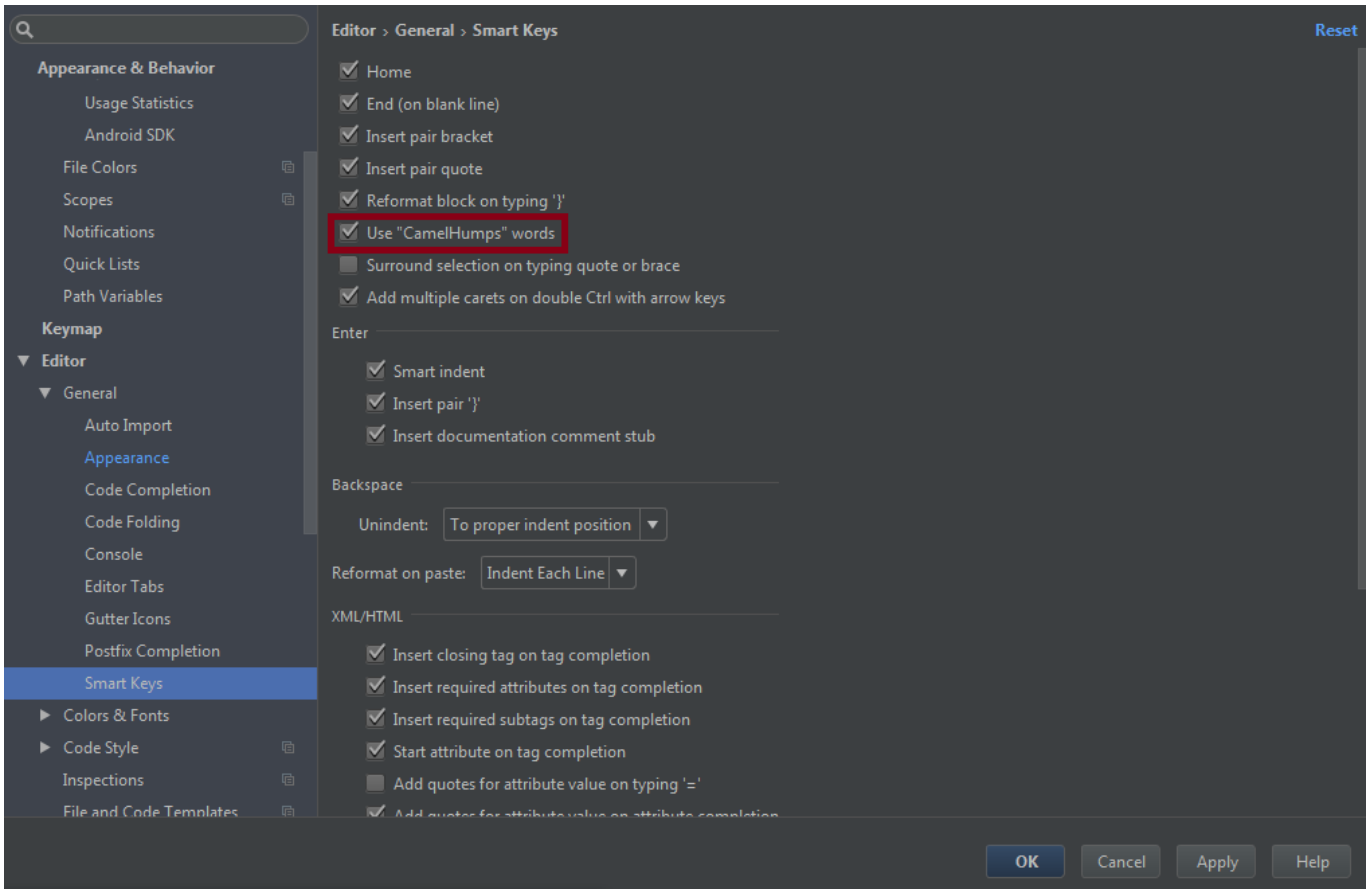
Por alguna razón, que no entiendo del todo, por defecto Android Studio no nos muestra la lineas dentro del editor.

Para habilitar esta funcionalidad deberemos acceder a ***File->Settings ->Editor->General->Appearance*** **y habilitar «*Show line numbers*».**

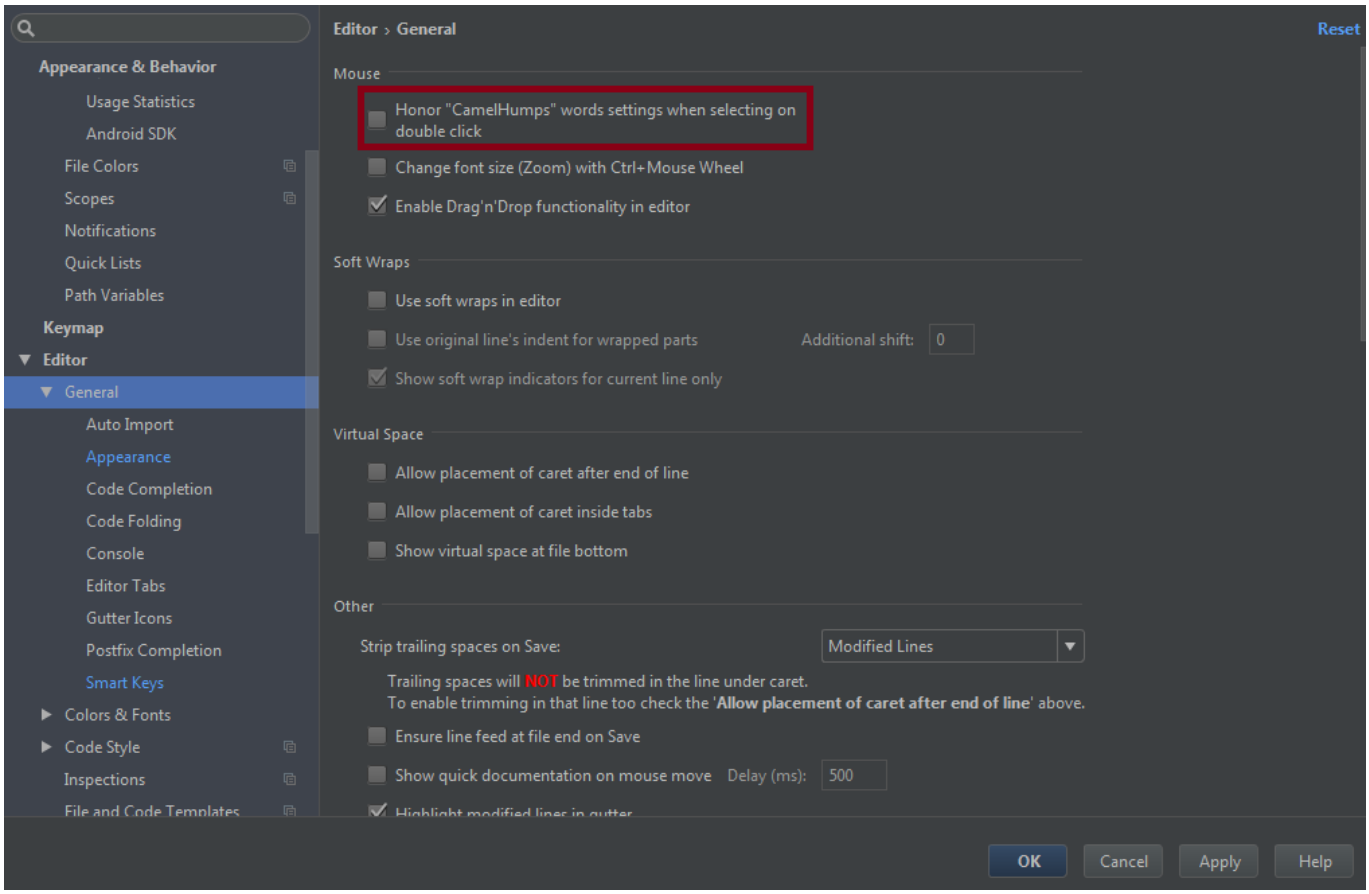


Camel humps

Android Studio no respeta las palabras ‘camel Humps’ cuando se navega a través del código con las teclas Ctrl + Izquierda / Derecha. Para solucionarlo accedemos a ***File->Settings->Editor->General->Smart Keys*** y finalmente *seleccionamos* ***Use ‘Camel Humps’ words***

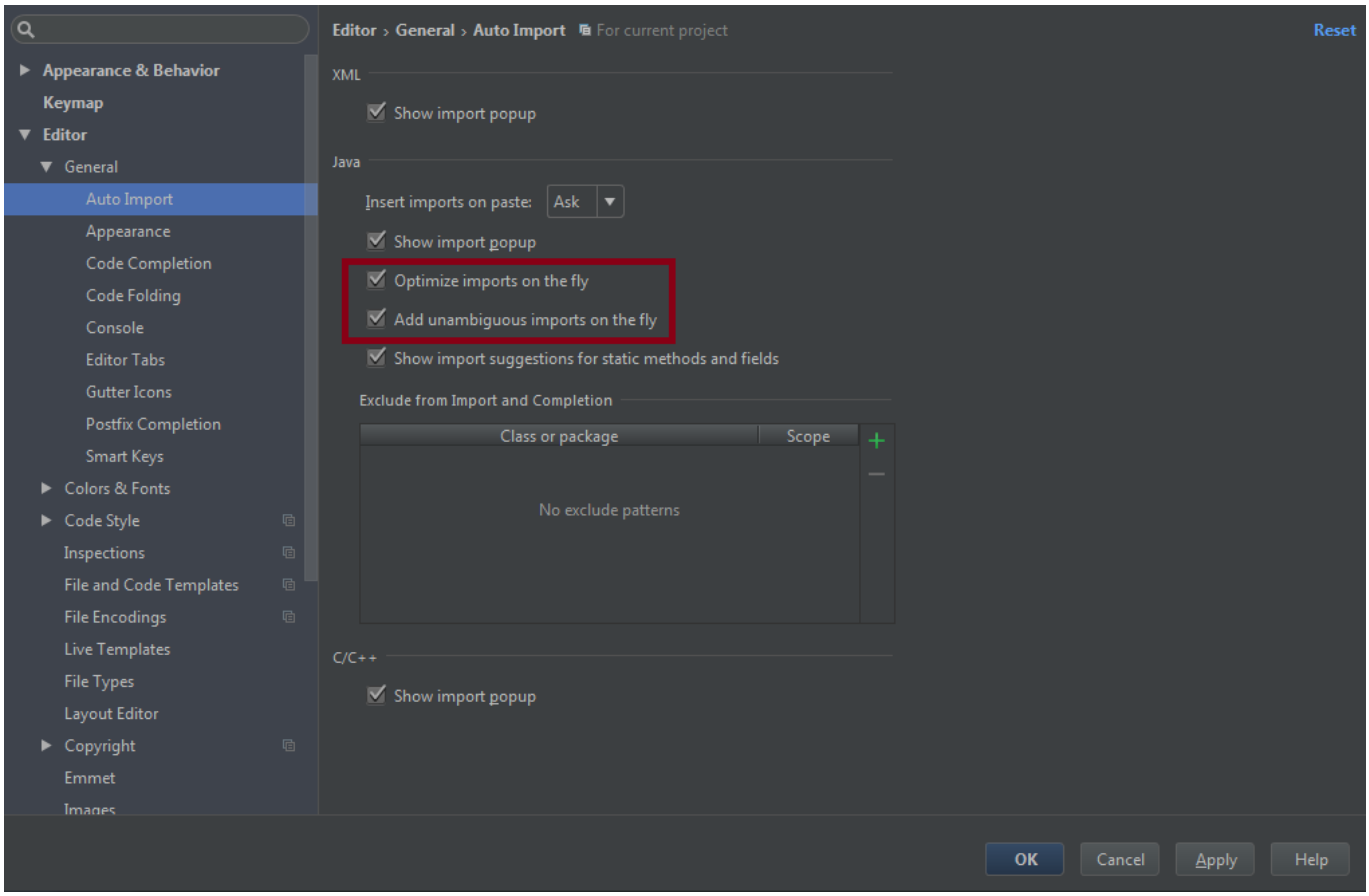


A continuación, lo que nos ocurrirá es que no podremos seleccionar con doble click una palabra. Para solucionarlo debemos acceder a ***File->Settings->Editor->General*** y deseleccionar ***Honor 'Camel Humps' words settings when selecting on double click***



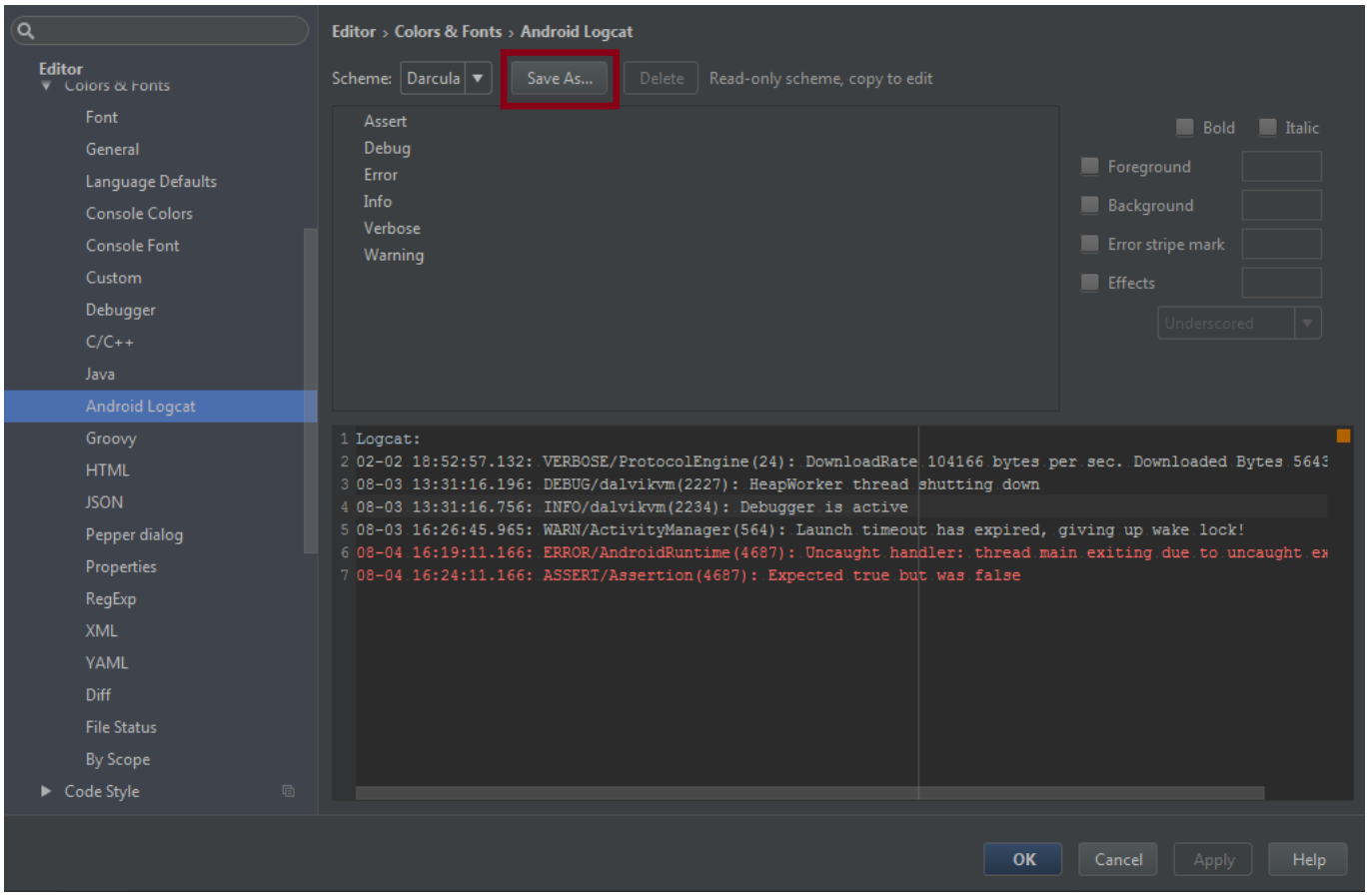
Imports Automáticos

Bien la verdad, es que hoy en día no es necesario saberse todos los packages que debemos usar, pues con una combinación de teclas podemos importarlos. Pero, ¿por que no hacerlo automáticamente? Para activarlo deberemos acceder a ***File->Settings->Editor->General->Auto Import***, seguidamente deberemos seleccionar ***Optimize imports on the fly*** y ***Add unambiguous imports on the fly***



Colores en el logview de Android Studio

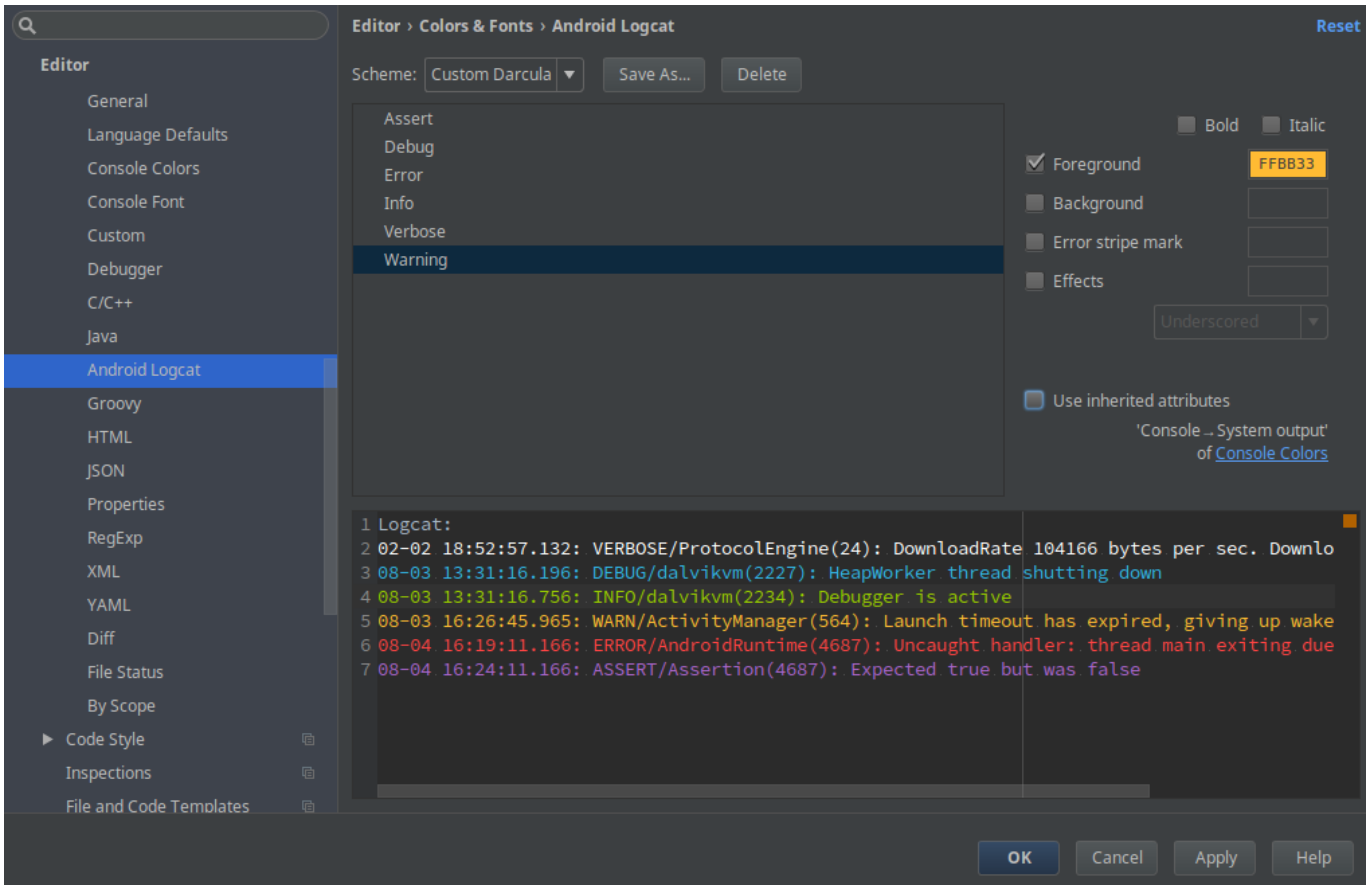
Bien esto es solo una preferencia. En logcat un buen esquema de colores nos ayudará a visualizar mejor la información. Deberemos acceder a ***File|Settings->Editor->Colors & Fonts->Android Logcat*** en este punto debemos hacer clic en ***Save As…*** y crear un nuevo esquema de colores.



Seguidamente por cada unos de los tipos de log (Assert, Debug, Error…) debemos deseleccionar ***‘Use inherited attributes’*** para cada color y aplicar el nuevo.

- Assert: #AA66CC
- Debug: #33B5E5
- Error: #FF4444
- Info: #99CC00
- Verbose: #FFFFFF
- Warning: #FFBB33

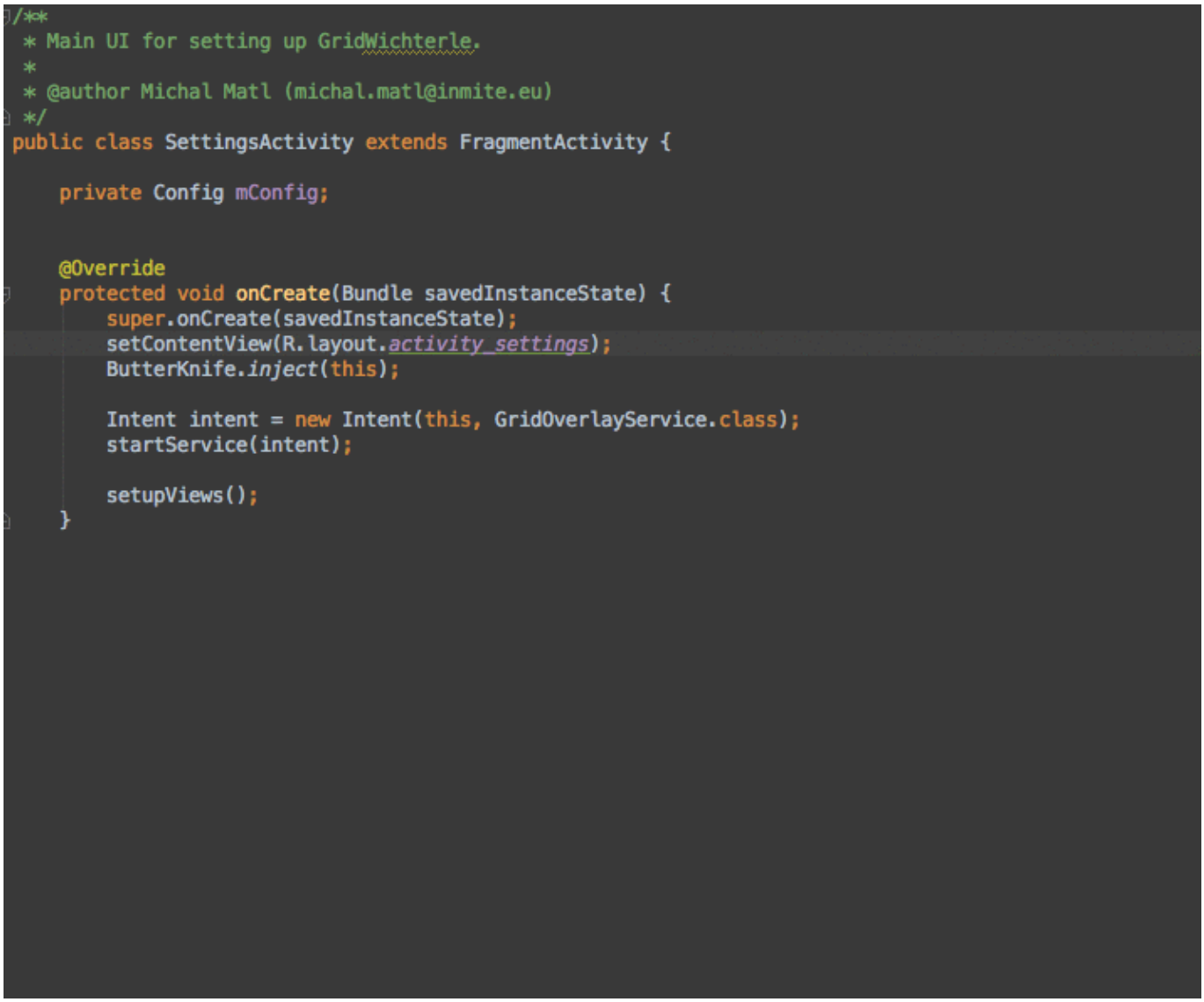
Este es el resultado



Plugins

Plugin ButterKnife Zelezny

De este plugin ya he hablado, la verdad es que nos facilita mucho el uso de Butterknife.Para instalarlo debemos acceder a ***File->settings->Plugin***, seguidamente clicar en ***Browse Repositories*** buscar ***ButterKnife Zelezny*** instalar y reiniciar. A continuación os dejo un gif de que muestra su utilización, sacado del proyecto de GitHub de este plugin. También os animo a que os paséis por la entrada de Butterknife



```java
/**
 * Main UI for setting up GridWichterle.
 *
 * @author Michal Matl (michal.matl@inmite.eu)
 */
public class SettingsActivity extends FragmentActivity {

    private Config mConfig;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);
        ButterKnife.inject(this);

        Intent intent = new Intent(this, GridOverlayService.class);
        startService(intent);

        setupViews();
    }
```

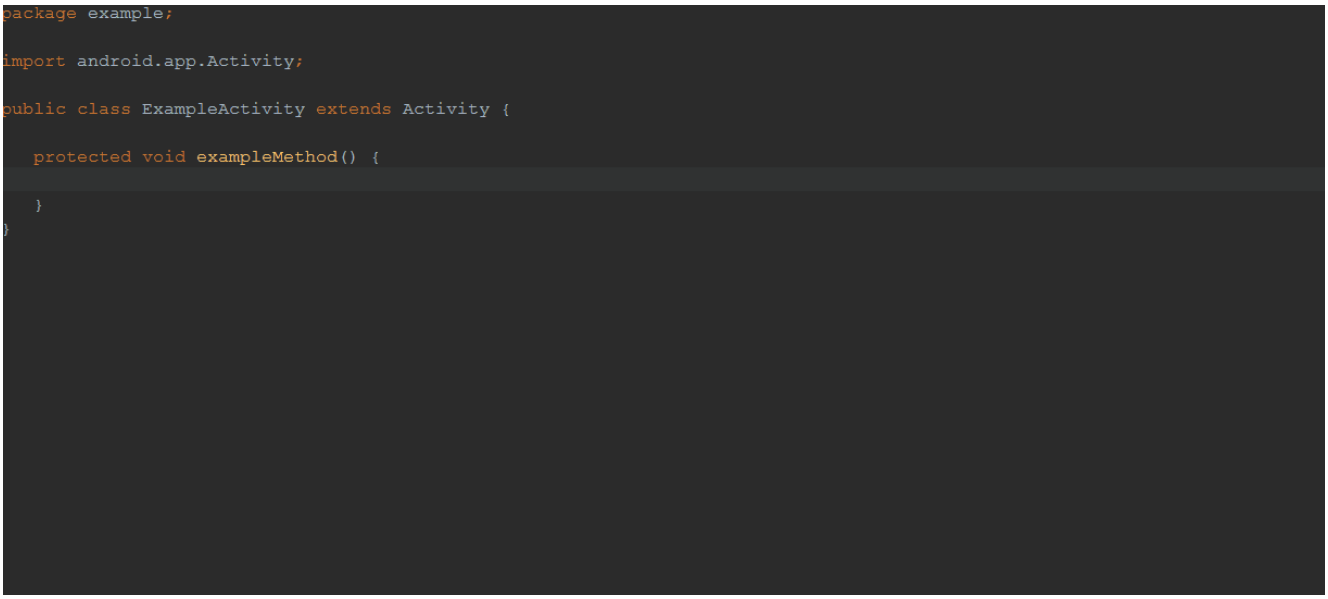
Plugin Exynap

Bien este plugin es espectacular. Nos permite encontrar e implementar el código que necesitemos en un instante. Para instalarlo debemos acceder a ***File->settings->Plugin***, seguidamente clicar en ***Browse Repositories*** buscar ***Exynap*** instalar y reiniciar.

Mediante la combinación de teclado ***Ctrl + Shift + D*** se abrirá una ventana contextual, des de la cual podremos buscar lo que necesitamos. Os dejo algunos gifs de su propia página.

```
package example;

import android.app.Activity;
import android.widget.TextView;

public class ExampleActivity extends Activity {

  protected void exampleMethod() {
    TextView textView = new TextView(this);
    textView
  }
}
```

```
package example;

import android.app.Activity;

public class ExampleActivity extends Activity {

  protected void exampleMethod() {
    int width
  }
}
```

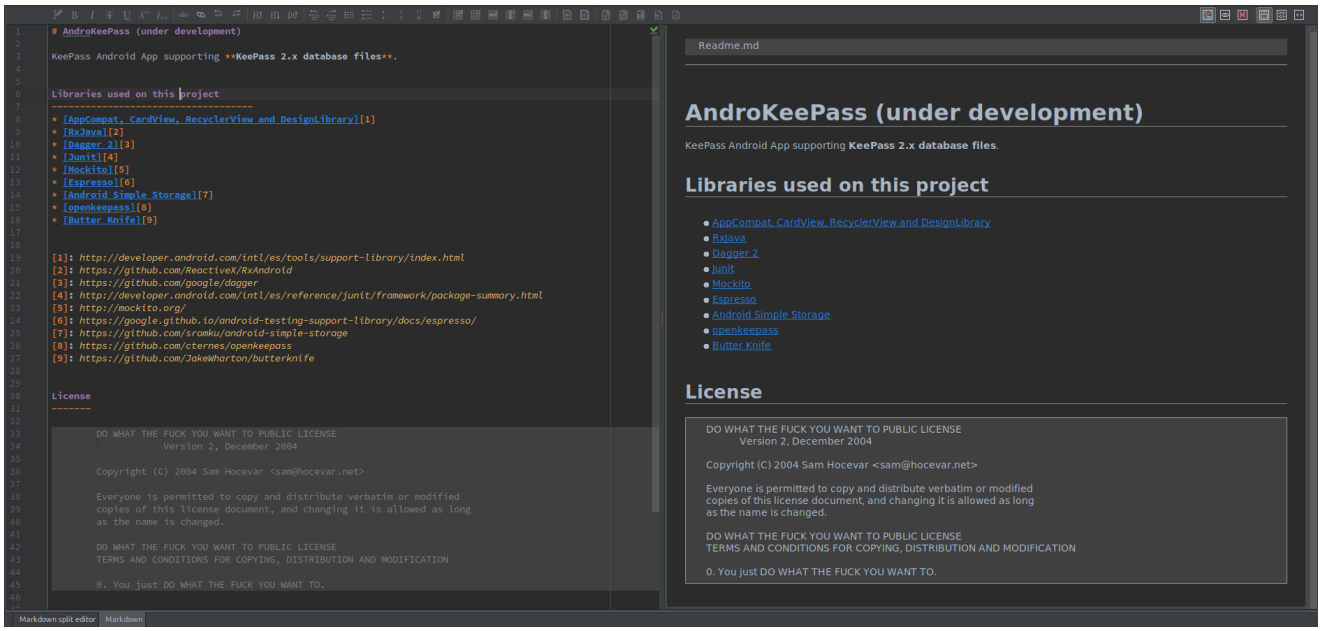


```
package example;

import android.app.Activity;

public class ExampleActivity extends Activity {

  protected void exampleMethod() {

  }
}
```

Markdown Navigator

Este plugin nos facilitara la edición de los archivos de markdown que tengamos en el proyecto. Por ejemplo ficheros de Changelog, Readme… Para instalarlo debemos acceder a ***File->settings->Plugin***, seguidamente clicar en ***Browse Repositories*** buscar ***Markdown Navigator*** instalar y reiniciar.

A continuación cuando abramos cualquier fichero de markdown se abrirá con el editor instalado.

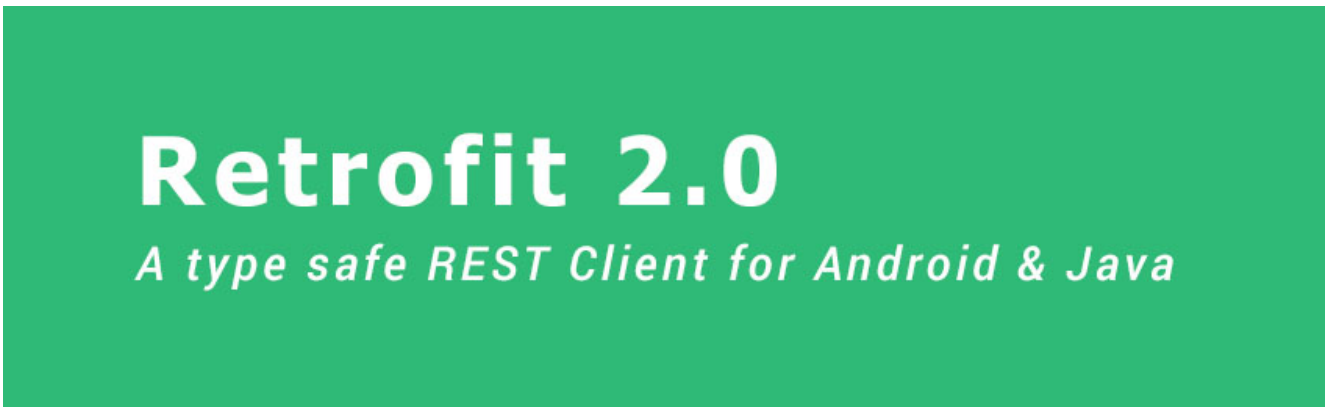


Observaciones

Bien, este post no ha sido de programación propiamente dicha. Ha sido una preconfiguración de Android Studio. La verdad es que estas pequeñas configuraciones y plugins nos ayudan a programar de manera más optima y rápida. En el siguiente post mostraremos atajos de teclado de Android Studio (que son varios y muy útiles).

Android – Retrofit y

Seeeduino Cloud



android – retrofit

Introducción

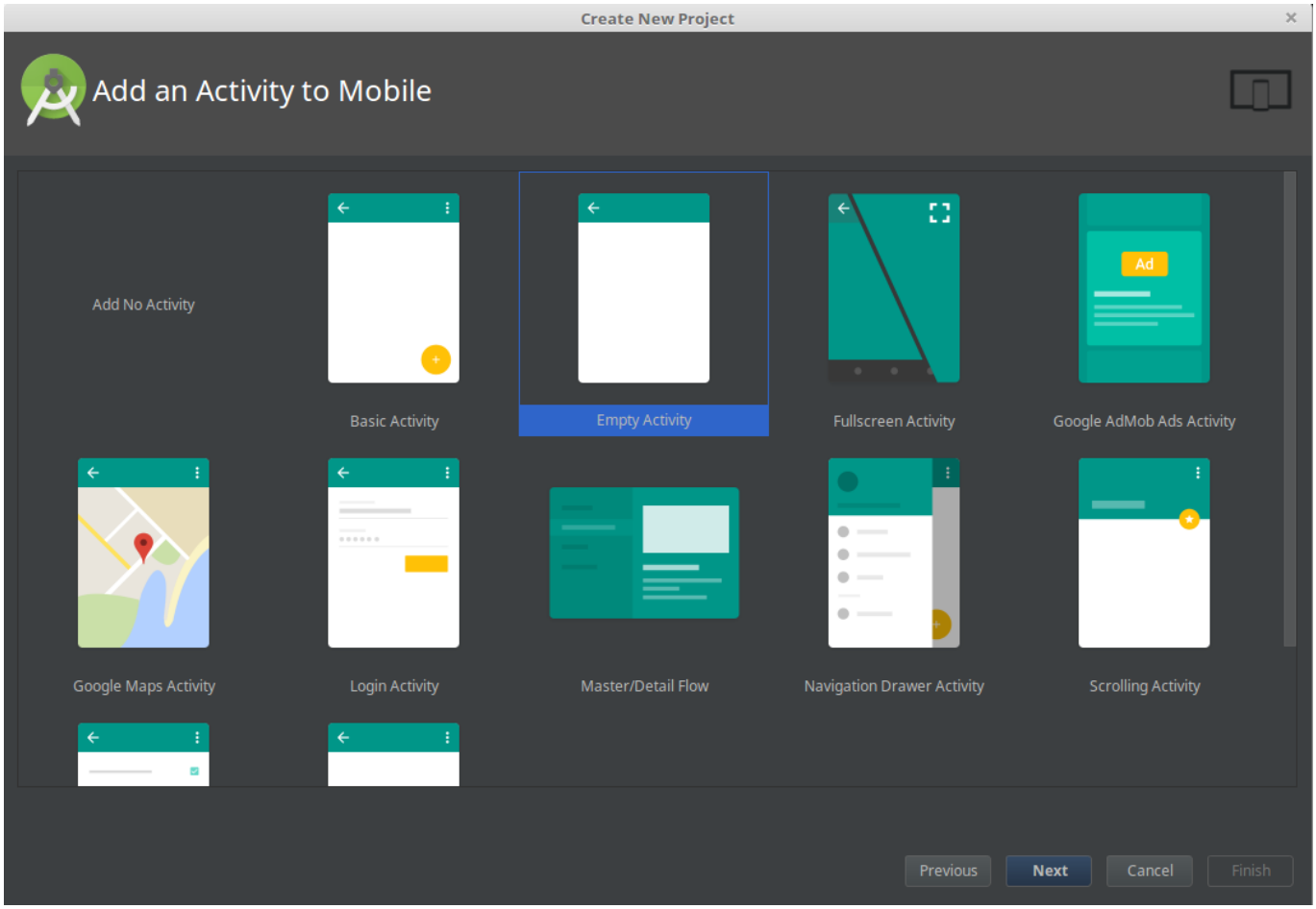
Y con este post llegamos a una de la librerías más útiles. Retrofit nos facilita el uso de llamadas rest, realizando las llamadas asíncronas sin que nosotros nos preocupemos de nada. Si además usamos GSON podemos obtener el resultado en una colección de objetos. La utilización de Retrofit en nuestros proyectos es muy fácil y ofrece un buen rendimiento. Una vez más, Square nos brinda una gran librería (ya hemos hablado anteriormente de Picasso).

Este post es una continuación de los de seeeduino (Seeduino Cloud – Parte 1 , Seeeduino Cloud – Parte 2). Al finalizarlo podremos controlar nuestro ventilador des de una aplicación Android.

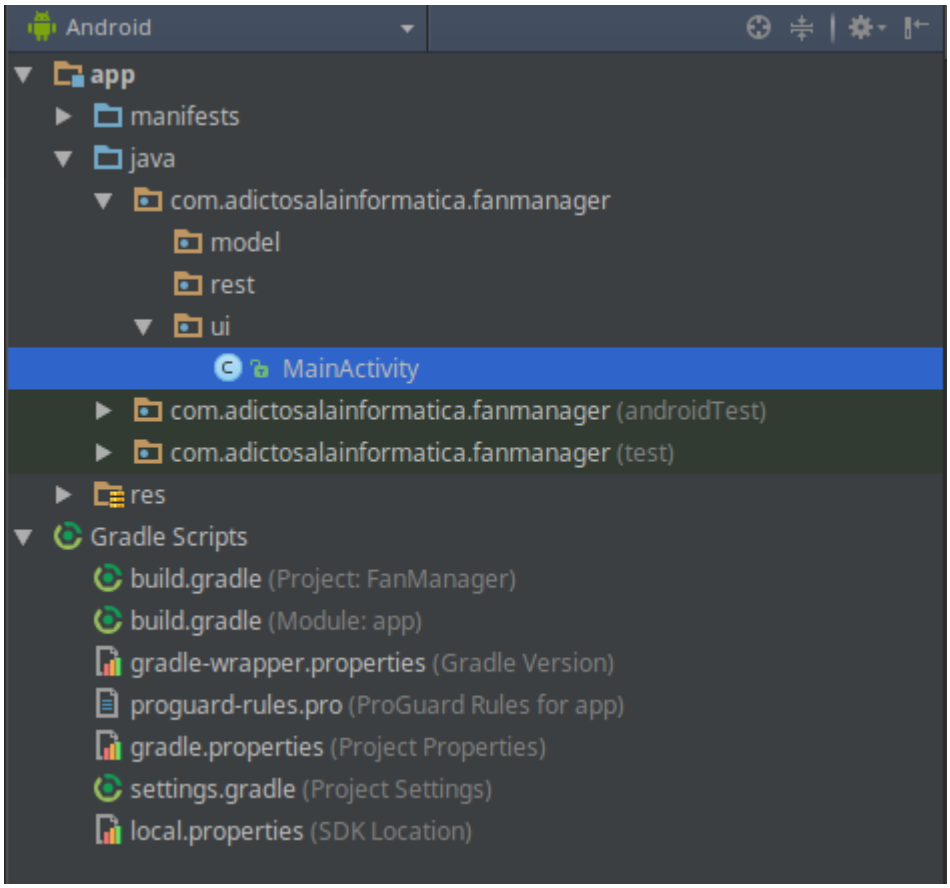
- Creando la estructura del proyecto
- Configuración
- Creando nuestras clases Pojo
- Creando una instancia de Retrofit
- Preparando end points
- Preparando nuestro activity

Creando el proyecto y su estructura

Crearemos un proyecto con un empty activity



A continuación crearemos una nueva estructura de packages. Esto es una costumbre, no es ningún estándar y cada developer utiliza la que más le conviene o resulta más fácil de usar, para organizar su código. Muy probablemente, cambiará durante el proceso desarrollo y aquí cada unos tiene sus preferencias. Lo que realmente es importante es utilizar una que nos ayude mantener nuestro código debidamente organizado.



Configuración

Añadiremos el plugin android-apt a nuestro classpath en el fichero **build.gradle**. Este se encuentra en la raíz de nuestro proyecto.

```
dependencies{
  classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
}
```

Dentro del archivo **app/build.gradle**, debemos añadir las dependencias de Retrofit.

```
apply plugin : 'com.neenbedankt.android-apt'

dependencies {

  // retrofit, gson
  compile 'com.google.code.gson:gson:2.6.2'
  compile 'com.squareup.retrofit2:retrofit:2.0.2'
  compile 'com.squareup.retrofit2:converter-gson:2.0.2'
```

```
  // picasso
  compile 'com.squareup.picasso:picasso:2.5.2'
  compile 'jp.wasabeef:picasso-transformations:2.1.0'

  // butterknife
  compile 'com.jakewharton:butterknife:8.4.0'
  apt 'com.jakewharton:butterknife-compiler:8.4.0'
}
```

Seguidamente añadiremos el permiso de red y de comprobación de esta a nuestro archivo **androidmanifest.xml**

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-
permission>
```

Creando nuestra clase Pojo

Esta es la clase base sobre la que Gson creará la instancia con los resultados de la llamada rest realizada con Retrofit, colocaremos la clase en el package **model**. Una de las maneras más fáciles de generar nuestras clases es utilizar un generador. Por ejemplo, jsonschema2pojo. Pero debemos tener cuidado, podemos encontrarnos fácilmente con una respuesta Json inmensa de la cual tan solo necesitamos alguna información concreta.

```
package com.adictosalainformatica.fanmanager.model;

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;

public class FanModel {

    @SerializedName("pin")
    @Expose
    private Integer pin;
    @SerializedName("status")
    @Expose
```

```
    private Integer status;

    /**
     *
     * @return
     * The pin
     */
    public Integer getPin() {
        return pin;
    }

    /**
     *
     * @param pin
     * The pin
     */
    public void setPin(Integer pin) {
        this.pin = pin;
    }

    /**
     *
     * @return
     * The status
     */
    public Integer getStatus() {
        return status;
    }

    /**
     *
     * @param status
     * The status
     */
    public void setStatus(Integer status) {
        this.status = status;
    }

}
```

Es importante no acabar con una colección de clases inmensa

llena de getters y setters que no vamos a utilizar. Todo ello para mantener nuestro código limpio o no acabar encontrándonos el error «64k method limit in dex». No es raro encontrarnos con él si utilizamos muchas librerías y colecciones de clases generadas a partir del resultado de una respuesta Json desmesurada. Podemos fácilmente solventar el problema revisando nuestras clases Pojo y eliminando todas aquellas variables y clases que no vamos a utilizar e incluso plantearnos si todas las librerías que estamos utilizando son realmente necesarias. Pero si esto no fuera suficiente, podemos solventar el problema con multidex a costa de perder la funcionalidad de instant run.

Creando una instancia de Retrofit

Para enviar solicitudes de red a una API, tenemos que utilizar la clase Retrofit.Builder y especificar la URL base para el servicio. Por lo tanto, crearemos una clase llamada ApiClient.java bajo en el package **rest**.

BASE_URL – es la url base de nuestra API. Vamos a utilizar esta url para todas las solicitudes posteriores.

```
package com.adictosalainformatica.fanmanager.rest;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;


public class ApiClient {

          public static final String BASE_URL =
"http://192.168.1.33/arduino/";
    private static Retrofit retrofit = null;


    public static Retrofit getClient() {
        if (retrofit==null) {
```

```
            retrofit = new Retrofit.Builder()
                    .baseUrl(BASE_URL)
.addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
}
```

Preparando end points

Los end points se definen dentro de una interfaz mediante anotaciones especiales de Retrofit para codificar información sobre los parámetros y el tipo de petición. Además, el valor de retorno es siempre una llamada con parámetros <T>, en nuestro caso <FanModel>. Crearemos la interface ApiInterface.java en el package **rest**

```
package com.adictosalainformatica.fanmanager.rest;

import
com.emotionexperience.fragancemanager.model.FraganceModel;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Path;


public interface ApiInterface {
    @GET("digital/{pin}/{value}")
     Call setPin(@Path("pin") int pin, @Path("value") int
value);

    @GET("status/{pin}")
    Call getStatus(@Path("pin") int pin);
}
```

Preparando nuestro activity

A continuación mostramos el código de nuestro activity que se encuentra en el package **ui**. Como se puede observar utilizamos Picasso y Butterknife

```
package com.adictosalainformatica.fanmanager.ui;

import android.app.ProgressDialog;
import android.content.Context;
import android.graphics.Color;
import android.net.ConnectivityManager;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.ImageButton;
import android.widget.Toast;

import com.adictosalainformatica.fanmanager.R;
import com.adictosalainformatica.fanmanager.model.FanModel;
import com.adictosalainformatica.fanmanager.rest.ApiClient;
import com.adictosalainformatica.fanmanager.rest.ApiInterface;
import com.squareup.picasso.Picasso;

import butterknife.BindView;
import butterknife.ButterKnife;
import butterknife.OnClick;
import
jp.wasabeef.picasso.transformations.ColorFilterTransformation;
import
jp.wasabeef.picasso.transformations.CropCircleTransformation;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class MainActivity extends AppCompatActivity {

    @BindView(R.id.main_btn_fan)
    ImageButton btnFan;

    // Constants
```

```
            private static final String TAG =
MainActivity.class.getName();
    private static int PIN = 8;

    private int fanStatus = 0;
    private static ApiInterface apiService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ButterKnife.bind(this);

        Picasso
                .with(getApplicationContext())
                .load(R.drawable.fan_image)
                .transform(new CropCircleTransformation())

                .into(btnFan);

                                        apiService     =
ApiClient.getClient().create(ApiInterface.class);

        if(isConetctionEnabled(getApplicationContext())){
            getFanStatus(PIN);
        }else{
             Toast.makeText(getApplicationContext(),"Internet
connection failed",Toast.LENGTH_LONG).show();
        }
    }

    @OnClick(R.id.main_btn_fan)
    public void switchFanStatus() {
        if(isConetctionEnabled(getApplicationContext())){
            if(fanStatus == 0){
                setFanStatus(PIN,1);
            }else{
                setFanStatus(PIN, 0);
            }
        }else{
```

```
            Toast.makeText(getApplicationContext(),"Internet
connection failed",Toast.LENGTH_LONG).show();
        }
    }

    /**
     * Get current Fan status
     * @param pin
     */
    private void getFanStatus(int pin) {
        Call <FanModel> call = apiService.getStatus(pin);
        call.enqueue(new Callback<FanModel>() {
            @Override
              public void onResponse(Call<FanModel> call,
Response<FanModel> response) {
                if (response.isSuccessful()) {
                    fanStatus =  response.body().getStatus();
                    Log.d(TAG, "Fan status: " + fanStatus);

                    if(fanStatus == 0){
                                              int  color  =
Color.parseColor("#33ee092b");
                        setColorFilter(color);

                    }else{
                                              int  color  =
Color.parseColor("#3300ff80");
                        setColorFilter(color);
                    }
                } else {
                    //request not successful (like 400,401,403
etc)
                    Log.e(TAG,response.message());
                }
            }

            @Override
               public void onFailure(Call<FanModel> call,
Throwable t) {
                // Log error here since request failed
                Log.e(TAG, "Error: " + t.toString());
```

```
            }
        });
    }

    /**
     * Set Fan status
     * @param pin
     */
    private void setFanStatus(int pin, int status){
        Call call = apiService.setPin(pin,status);
        call.enqueue(new Callback<FanModel>() {
            @Override
               public void onResponse(Call<FanModel> call,
Response<FanModel> response) {
                if (response.isSuccessful()) {
                    fanStatus =  response.body().getStatus();
                    Log.d(TAG, "Fan status: " + fanStatus);

                    if(fanStatus == 0){
                                                int  color  =
Color.parseColor("#33ee092b");
                        setColorFilter(color);

                    }else{
                                                int  color  =
Color.parseColor("#3300ff80");
                        setColorFilter(color);
                    }
                } else {
                    //request not successful (like 400,401,403
etc);
                    Log.e(TAG,response.message());
                }
            }

            @Override
               public void onFailure(Call<FanModel> call,
Throwable t) {
                // Log error here since request failed
                Log.e(TAG, "Error: " + t.toString());
            }
```

```
        });
    }

    /**
     * Set image filter color
     * @param color
     */
    private void setColorFilter(int color){
        Picasso
                .with(getApplicationContext())
                .load(R.drawable.fan_image)
                                            .transform(new
ColorFilterTransformation(color))
                .transform(new CropCircleTransformation())
                .into(btnFan);
    }

    /**
     * Tests if there's connection
     * @param cx context application
     * @return true or false
     */
    public static boolean isConetctionEnabled(Context cx){
                        ConnectivityManager   conMgr   =
(ConnectivityManager)cx.getSystemService(Context.CONNECTIVITY_
SERVICE);

        if (conMgr.getActiveNetworkInfo() != null
                && conMgr.getActiveNetworkInfo().isAvailable()
                                                        &&
conMgr.getActiveNetworkInfo().isConnected()) {
            return true;
        } else {
            return false;
        }
    }
}
```

I finalmente, nuestro layout

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

```
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.adictosalainformatica.fanmanager.ui.MainAct
ivity">


    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/fan_image"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:id="@+id/main_btn_fan"
        android:background="@null"
        android:padding="10dp"/>
</RelativeLayout>
```

Desde este enlace os podéis descargar fan_image

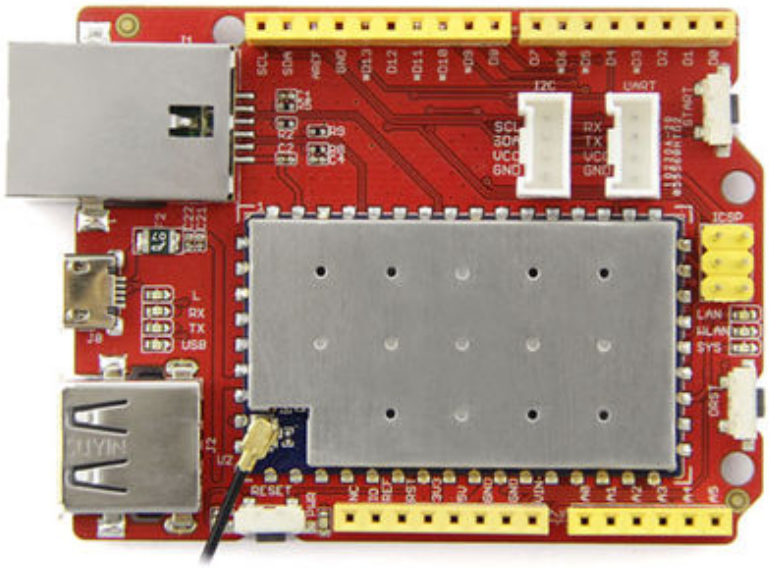
Observaciones

Después de mucho tiempo lidiando con AsyncTask, rotaciones, memory leaks… Retrofit nos permite abstraernos de todo esto y además es muy fácil de utilizar. Por todo ello, Retrofit se convierte en una librería casi indispensable. Finalmente, dejo el enlace a la web de Retrofit y un ejemplo en Github. El cual, nos permite apagar y encender un ventilador siempre y cuando tengamos nuestro Seeduino Cloud configurado y preparado para trabajar con un relayshield.

- Retrofit
- Seeeduino Cloud – Parte 1 (configuración Seeduino Cloud)
- Seeeduino Cloud – Parte 2 (Montaje relayshield)

- FanManager

Seeeduino Cloud – Parte 2



Seeeduino Cloud

Introducción

En este post añadiremos una relay shield a nuestro seeeduino. Seguidamente, realizaremos el cableado con uno de los relay y conectaremos un pequeño ventilador. De esta manera podremos encender y apagar nuestro ventilador remotamente.

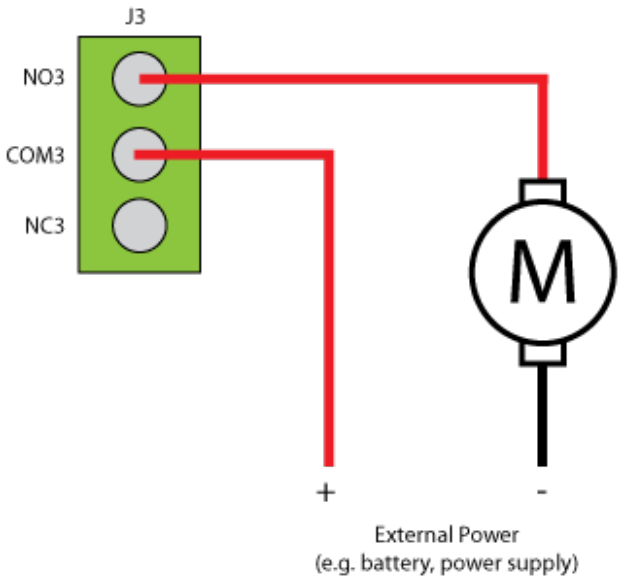
- Montando el Relay Shield y cableado de un relay
- Programa para controlar el relay
- Encendiendo y apagando nuestro ventilador

Montando el Relay Shield y cableando un relay

El funcionamiento de un relay es bastante simple. Un relay es un switch electromagnético. Un relay tiene tres entradas:

- **nc** (normally closed). El circuito tiene corriente, a no ser que se active el relay.
- **no** (normally open). El circuito no tiene corriente, a no ser que se active el relay.
- **com** (entrada de electricidad). Punto de entrada del corriente eléctrico externo.

En el siguiente ejemplo podemos ver la conexión de un relay con un motor. Si observamos el diagrama el motor no estará en funcionamiento hasta que se active el relay.

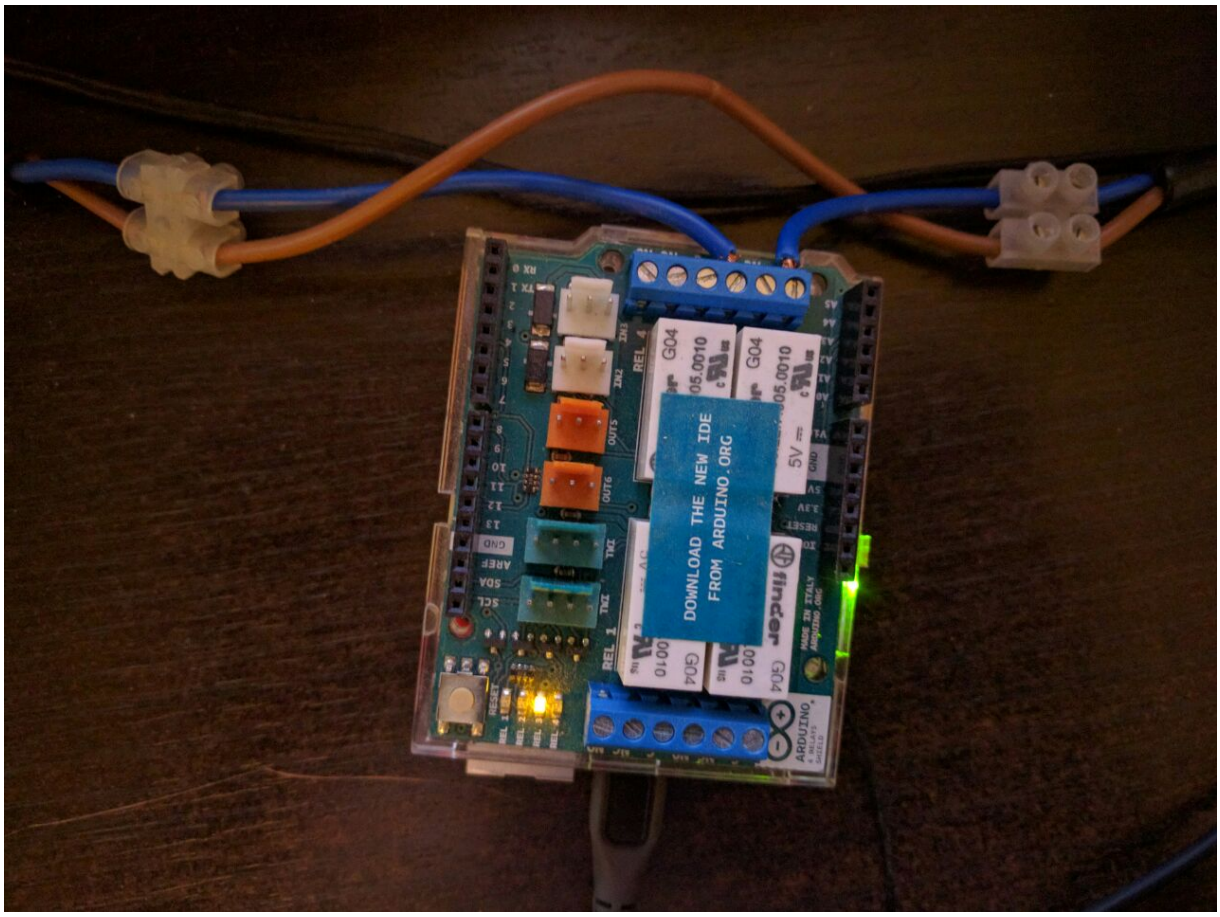


Motor-shield-schematic-drawing

Primeramente montaremos la relay shield encima de nuestro Seeeduino (Encaja perfectamente).

Seguidamente cablearemos un relay. En este ejemplo utilizaremos el relay 3 (digital pin 8), pero se podría utilizar cualquiera. La entrada de corriente estará en el conector **C**

, y la salida en el conector **NO**



Seguidamente la embra:

Y finalmente la toma de electricidad:

Solo nos quedará conectar el macho a la corriente y un ventilador a la hembra.

Para más información sobre el relay shield utilizado os dejo el link de la página del mismo Arduino – 4 Relays Shield

Programa para controlar el relay

A continuación este es el programa que permitirá acceder remotamente a nuestro relay. El código esta debidamente comentado

```
#include <Bridge.h>
```

```
#include <BridgeServer.h>
#include <BridgeClient.h>

// Listen to the default port 5555, the Yún webserver
// will forward there all the HTTP requests you send
BridgeServer server;

int RELAY3 = 8;

void setup() {
  // Bridge startup
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();
  digitalWrite(13, HIGH);
  pinMode(RELAY3, OUTPUT);

  // Listen for incoming connection only from localhost
  // (no one from the external network could connect)
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  // Get clients coming from server
  BridgeClient client = server.accept();

  // There is a new client?
  if (client) {
    // Process request
    process(client);

    // Close connection and free resources.
    client.stop();
  }

  delay(50); // Poll every 50ms
}

void process(BridgeClient client) {
  // read the command
```

```
  String command = client.readStringUntil('/');

  // is "digital" command?
  if (command == "digital") {
    digitalCommand(client);
  }

  // is "status" command?
  if (command == "status") {
    statusCommand(client);
  }
}

void digitalCommand(BridgeClient client) {
  int pin, value;

  // Read pin number
  pin = client.parseInt();

  // If the next character is a '/' it means we have an URL
  // with a value like: "/digital/13/1"
  if (client.read() == '/') {
    value = client.parseInt();
    digitalWrite(pin, value);
  } else {
    value = digitalRead(pin);
  }

  // Send feedback to client
  client.print(F("{\"pin\":"));
  client.print(pin);
  client.print(F(" ,\"status\":"));
  client.print(value);
  client.print(F("}"));

  // Update datastore key with the current pin value
  String key = "D";
  key += pin;
  Bridge.put(key, String(value));

}
```

```
void statusCommand(BridgeClient client) {
  int pin, value;

  // Read pin number
  pin = client.parseInt();
  value = digitalRead(pin);

  // Send feedback to client
  client.print(F("{\"pin\":"));
  client.print(pin);
  client.print(F(" ,\"status\":"));
  client.print(value);
  client.print(F("}"));
}
```

Encendiendo y apagando nuestro ventilador

Desde un navegador accediendo a la url obtendremos el estado del relay:

http://ip-seeduino/arduino/status/8

{"pin":8 ,"status":0}

Retornará un 0 (apagado) o 1 (encendido)

Para apagarlo

http://ip-seeeduino/arduino/digital/8/0

{"pin":8 ,"status":0}

Y para encenderlo

http://ip-seeeduino/arduino/digital/8/1

{"pin":8 ,"status":1}

Conclusión

Ya podemos controlar nuestro ventilador remotamente y como

podemos ver ha resultado muy fácil. Pero la verdad, con una url desde el plugin de un navegador es muy rudimentario y no excesivamente útil. En el próximo post crearemos una simple aplicación Android, que se encargará de controlar y obtener el estado de nuestro ventilador des de la Rest Api de Seeeduino. De paso presentaremos y utilizaremos una librería muy útil para este tipo de escenarios, Retrofit. A continucación podéis encontrar el ejemplo del código arduino en GitHub y el enlace a la placa de rayls de arduino.

- Arduino 4 Relay Shield
- Seeeduino Rest Api example

Android – Butterknife

Introducción

En este post trataremos con una librería muy útil. En este caso nos ayuda deshacernos de mucho código. De esta manera nuestra aplicación nos queda mucha más limpia y legible. Lo mejor de todo es que esta librería inyecta código en tiempo de compilación, es decir, el rendimiento de nuestra aplicación no se verá afectado por su uso.

- Configuración
- Ejemplo de uso con Activity
- Ejemplo de uso con Fragment
- Eventos
- Usando Resources
- Plugin ButterKnife Zelezny

Configuración

Para la instalación, tenemos que añadir el plugin android-apt a nuestro classpath en el fichero build.gradle. Este se encuentra en la raíz de nuestro proyecto.

```
dependencies{
  classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
}
```

Dentro del archivo app/build.gradle, debemos añadir el plugin antes de añadir las dependencias de Butterknife.

```
apply plugin : 'com.neenbedankt.android-apt'

dependencies {
  compile 'com.jakewharton:butterknife:8.0.1'
  apt 'com.jakewharton:butterknife-compiler:8.0.1'
}
```

Ejemplo de uso

Eliminaremos el uso de findViewById utilizando @BindView en

los views de Android (TextView, Button, EditText…):

```
class MainActivity extends Activity {
  // Automatically finds each field by the specified ID.
  @BindView(R.id.title) TextView title;
  @BindView(R.id.name) EditText name;
  @BindView(R.id.btn_send_name) Button sendName;

  @Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
    ButterKnife.bind(this);
    // Some code
  }
}
```

Como podemos ver se simplifica el código y a su vez se hace más legible.

Ejemplo de uso con Fragment

Cuando utilicemos fragments tendremos que especificar en el método bind la vista con la que vamos a trabajar y en el evento onDestroyView utilizar unbind:

```
public class SimpleFragment extends Fragment {
  @BindView(R.id.txt_name) Button name;
  @BindView(R.id.btn_send_name) Button sendName;

  @Override public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
     View view = inflater.inflate(R.layout.simple_fragment,
container, false);
    ButterKnife.bind(this, view);
    // Some code
    return view;
  }

  // When binding a fragment in onCreateView, set the views to
null in onDestroyView.
   // Butter Knife has an unbind method to do this
```

```
automatically.
  @Override public void onDestroyView() {
    super.onDestroyView();
    ButterKnife.unbind(this);
  }
}
```

Eventos

Los eventos serán funciones con la anotación correspondiente.

```
@OnClick(R.id.submit)
public void sayHi(Button button) {
  button.setText("Hello!");
}
```

También podemos agrupar vistas y asignarlas a un único evento

```
@OnClick({R.id.maint_btn_change_text,
R.id.main_btn_new_intent})
    void buttonClick(View v) {
        switch (v.getId()){
            case R.id.maint_btn_change_text:
                title.setText(name.getText().toString());
                break;
            case R.id.main_btn_new_intent:
                 Intent resourcesIntent =  new Intent(this,
ResourcesActivity.class);
                startActivity(resourcesIntent);
                break;
        }
    }
```

Podremos utilizar los siguientes eventos: OnClick, OnLongClick, OnEditorAction, OnFocusChange, OnItemClick, OnItemLongClick,OnItemSelected, OnPageChange, OnTextChanged, OnTouch, OnCheckedChanged.

Usando resources

Podemos hacer binding de resources fácilmente

```
@BindString(R.string.title) String title;
@BindDrawable(R.drawable.my_drawable) Drawable myDrawable;
@BindColor(R.color.red) int red;
```

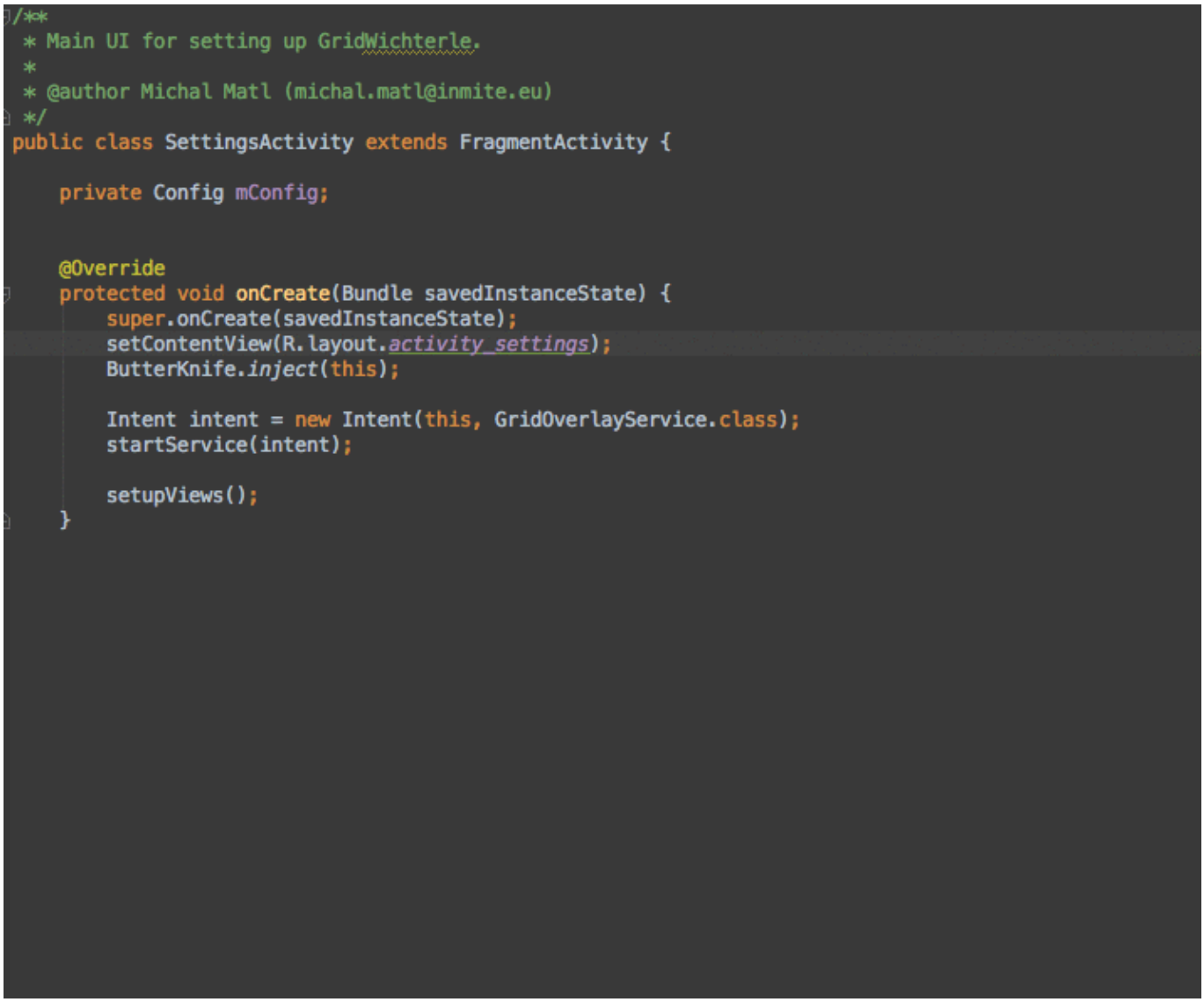
Plugin ButterKnife Zelezny

Este es un plugin para Android Studio muy útil. A partir de una vista nos genera todos los bindviews para esta.

Primeramente deberemos instalar el plugin en Android Studio:

- Des de Android Studio: iremos a **File** -> **Settings** -> **Plugins** -> **Browse repositories** y buscaremos **ButterKnife Zelezny**
- Descargandolo: decargamos el plugin ButterKnife Zelezny y lo instalamos des de **File** -> **Settings** -> **Plugins** -> **Install plugin from disk**

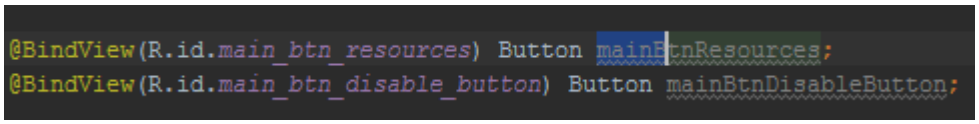
Para hacer uso del plugin debemos incluir Butterknife tal y como hemos mostrado al principio del post. Finalmente, os dejo un gif sacado de la página del proyecto de GitHub donde se muestra su utilización.



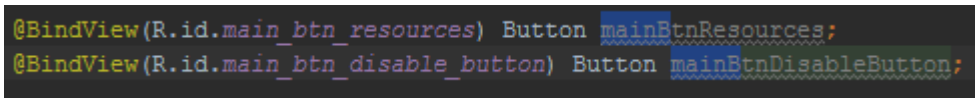
```java
/**
 * Main UI for setting up GridWichterle.
 *
 * @author Michal Matl (michal.matl@inmite.eu)
 */
public class SettingsActivity extends FragmentActivity {

    private Config mConfig;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);
        ButterKnife.inject(this);

        Intent intent = new Intent(this, GridOverlayService.class);
        startService(intent);

        setupViews();
    }
```

Como nota final, decir que se generan los nombres de las variables a partir de los ids asignados a las vistas del layout seleccionado. Es decir, **main_btn_show_toast** se convertirá en **mainBtnShowToast**. En el cuadro de dialogo podremos modificar esos nombres, uno a uno claro. Si observamos atentamente un correcto «naming» en nuestro layout nos generará automáticamente variables con un «naming» adecuado. El problema es que nos encontramos con la anotación correspondiente al layout, **main**BtnShowToast. Solucionar esto es fácil, cogeremos como ejemplo las variables correspondientes a los Buttons del layout main:

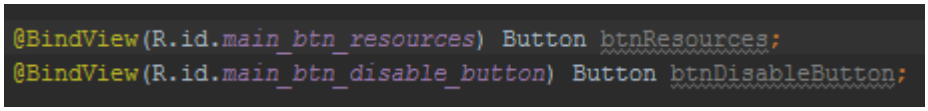
- Seleccionamos el layout y la primera letra correspondiente a la vista que queremos modificar



- Seguidamente con atajo de teclado **alt + j** seleccionaremos todas la variables correspondientes al layout y la inicial de la vista a modificar



- Y finalmente, corregimos el nombre para todas las variables

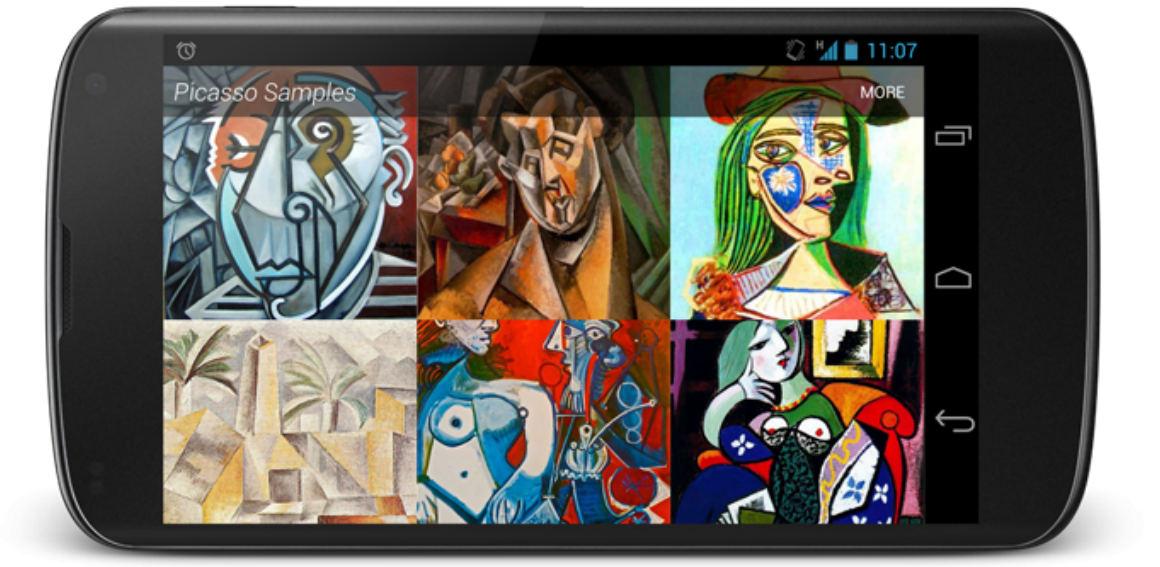


Observaciones

Butterknife nos ofrece una manera de mantener nuestro código limpio y legible. Algo que a largo plazo se hace indispensable, pues cualquier software se va a tener que mantener. Y que nuestro código sea limpio y legible es vital para esta tarea. Hemos mostrado también un plugin muy útil para utilizar la librería y al final hemos hablado un poco de naming. Esto daría para otro post y llegará. Por otro lado también hemos mostrado un atajo de teclado de Android Studio, los atajos de teclado para Android Studio dan para otro futuro post y así podríamos seguir y no parar nunca. Finalmente, dejo el enlace a la web de Butterknife (donde podréis encontrar ejemplos más avanzados), el enlace a la cuenta de GitHub del plugin ButterKnife Zelezny y un simple ejemplo en Github (el proyecto de Picasso refactorizado usando Butterknife).

- Butterknife
- ButterKnife Zelezny
- ButterKnifeTest

Android – Picasso



Introducción

Seguimos con una de esas librerías que nos solucionan de manera espectacular la utilización de imágenes en nuestros proyectos, Picasso. Existen otras como Glide (recomendada por Google) o Fresco (Facebook). Las diferencias entre Picasso y Glide son pocas, en cuanto a Fresco se refiere, es una aproximación diferente al tratamiento de imágenes en android. En este post (nearsoft) podeis ver una breve comparación entre la tres y en este otro post (inthecheesefactory blog) encontraras una comparación entre Glide y Picasso. Bien, vamos con Picasso:

- Configuración y utilización
- Picasso con assets o drawables
- Resize ,fit y rotation
- Scaling
- Placeholder y error
- Picasso Transformation Library

- Cache Indicators y Logging
- Observaciones

Configuración y utilización

Dentro del archivo app/build.gradle, debemos añadir la dependencia de Picasso.

```
dependencies{
  compile 'com.squareup.picasso:picasso:2.5.2'
}
```

Una vez añadida podemos empezar a utilizarla, como podéis ver es muy simple:

```
ImageView ourImageView = (ImageView) findViewById(R.id.imageView);
String remoteUrl = "http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg";

Picasso
    .with(this)
    .load(remoteUrl)
    .into(ourImageView);
```

Picasso con assets, drawables o ficheros guardados

Realmente fácil:

```
// Loading drawable
Picasso
    .with(this)
    .load(R.drawable.image)
    .into(imageView1);

// Loading asset
Picasso
    .with(this)
    .load("file:///android_asset/image.png")
```

```
    .into(imageView2);

// Loading file from storage
File file = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), "Android.png");
Picasso
    .with(this)
    .load(file)
    .into(imageView3);
```

Resize, fit y rotation

Podemos redimensionar la imagen.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg")
    .resize(100, 100)
    .into(imageView);
```

Pero en caso de que la imagen sea más pequeña tenemos la opción de evitar este reescalado. Redimensionar una imagen haciéndola más grande nos supone un uso de recursos que muchas veces nos va a dar un resultado muy pobre. En este caso podemos utilizar **scaleDown(true)**, de esta manera la imagen se redimensionará si el resultado final implica unas dimensiones inferiores a las originales.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg")
    .resize(100, 100)
    .scaleDown()
    .into(imageView);
```

O bien hacer que se redimensione automáticamente al tamaño del imageView. Esto puede hacer que la carga de la imagen tarde un

poco más, puesto que primero se debe esperar a poder obtener el tamaño del imageView.

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .fit()
    .into(imageView);
```

Podemos rotar la imagen

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .rotate(180f)
    .into(imageView);
```

E incluso indicar que punto queremos utilizar para pivotar la rotación

```
// rotate(float degrees, float pivotX, float pivotY)
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .rotate(45f, 200f, 100f)
    .into(imageViewComplexRotate);
```

Scaling

El reescalado de imágenes puede afectar el aspect ratio y hacer que esta se vea deforme. Para solucionar esto podemos utilizar centerCrop() o centerInside().

CenterCrop
Se escala la imagen haciendo que coincida con los límites del ImageView y entonces se elimina la parte restante de la imagen. El ImageView contendrá la imagen pero seguramente se perderán partes de esta.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .resize(100, 100) // resizes the image to these dimensions
(in pixel)
    .centerCrop()
    .into(imageViewResizeCenterCrop);
```

CenterInside

Se escala la imagen teniendo en cuenta que las dos dimensiones de la imagen son iguales o inferiores al tamaño del imageView. La imagen se mostrara completa pero puede que no ocupe todo el imageView.

```
Picasso
    .with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .resize(100, 100)
    .centerInside()
    .into(imageViewResizeCenterInside);
```

Placeholder y error

Podemos utilizar una imagen temporal hasta que nuestra imagen se haya cargado y otra en caso de que ocurra un error. De esta manera, el usuario tendrá la sensación que durante un tiempo de espera o incluso un error la aplicación funciona perfectamente.

```
Picasso.with(this)
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .placeholder(R.drawable.placeholder_image)
    .error(R.drawable.error_image)
    .into(imageView);
```

Picasso Transformation Library

Existe una librería que nos permite hacer transformaciones a nivel avanzado y de manera muy fácil picasso-transformations. Os recomiendo que paséis por su cuenta de Github puesto que aquí solo mostraremos dos ejemplos.

Primeramente en el archivo app/build.gradle, debemos añadir la dependencia de picasso-transformations.

```
dependencies{
    compile 'jp.wasabeef:picasso-transformations:2.1.0'
}
```

Una vez añadida podemos empezar a tulizarla, por ejemplo para aplicar un crop circular:

```
    Picasso
    .with(this)

.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .transform(new CropCircleTransformation())
    .into(imageView);
```

También podemos encadenar transformaciones, por ejemplo haciendo un crop como en el ejemplo anterior y aplicando un filtro de color:

```
int color = Color.parseColor("#3300ff80");
Picasso
    .with(this)
    .load("http://www.guitarthai.com/picpost/gtpicpost/Q367224
.jpg")
    .transform(new ColorFilterTransformation(color))
    .transform(new CropCircleTransformation())
    .into(imageView);
```

Cache Indicators y Logging

Cache Indicators

Picasso utiliza dos tipos de chache, memoria y almacenamiento. En algunos casos, para comprobar y hacer tests de rendimiento de nuestra aplicación, nos interesará saber de donde se ha obtenido la imagen. Para hacerlo debemos añadir la opción **.setIndicatorsEnabled(true)**

```
Picasso
    .with(this)
    .setIndicatorsEnabled(true);
Picasso
    .with(this)
    .setIndicatorsEnabled(true);
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .into(imageView);
```

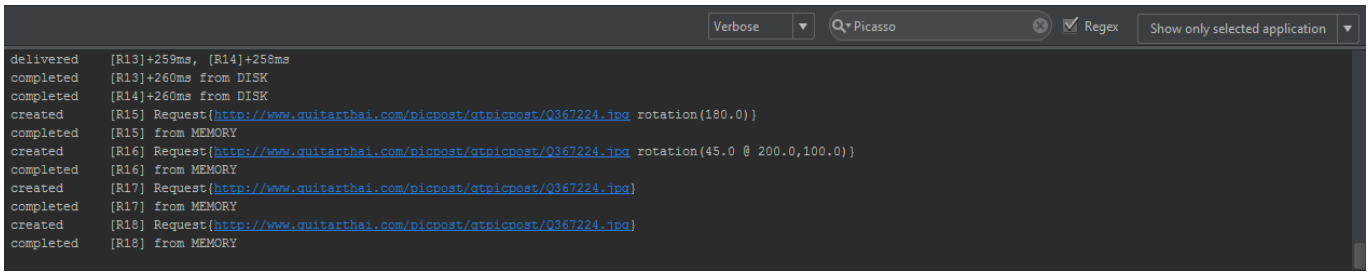
Una vez cargadas la imágenes estas tendrán un indicador de color en la parte superior izquierda. El esquema de colores corresponde al origen del cual la imagen es:

- Verde (memoria, mejor rendimiento)
- Azul (memoria interna del dispositivo, rendimiento medio)
- Rojo (red, el peor rendimiento)

Logging

Aveces necesitamos más información que una simple indicación del origen de la imagen. Utilizando la opción **.setLoggingEnabled(true);** obtendremos en el log una información más acurda del proceso creado por Picasso.

```
Picasso
    .with(this)
    .setLoggingEnabled(true);
Picasso
    .with(this)
    .setLoggingEnabled(true);
.load("http://www.guitarthai.com/picpost/gtpicpost/Q367224.jpg
")
    .into(imageView);
```

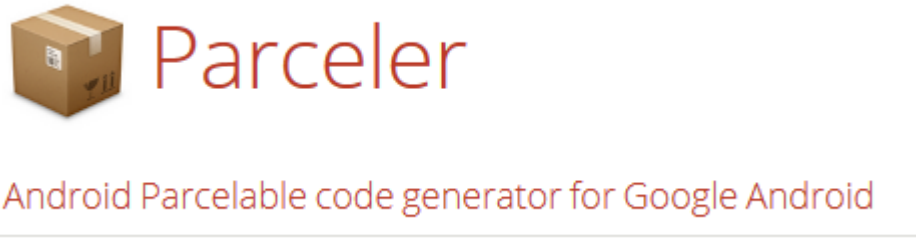


Observaciones

Bien, hoy hemos tratado una librería la cual conviene tener en la caja de herramientas. Nos ofrece muchas opciones y nos permite realizar operaciones de una manera muy simple. Como contrapartida podemos decir que el rendimiento siempre se verá afectado. Finalmente, dejo el enlace a la web de Picasso y un simple ejemplo en Github.

- Picasso
- PicassoTest

Android – Parceler



Introducción

Como se comento en el último post de Java para android en adelante trataremos librerías y ejemplos de novedades, para cursos de android podéis encontrar enlaces en el post

anterior. En dicho post en la parte de serialización comentábamos que en android existían parcelables y el porqué debíamos utilizarlos. Bien pues hoy presentamos una librería que facilita mucho su uso Parceler

- Configuración
- Utilizando Parceler en nuestra clase
- Observaciones

Configuración

Para la instalación, tenemos que añadir el plugin android-apt a nuestro classpath en el fichero build.gradle que se encuentra en la raíz de nuestro proyecto.

```
dependencies{
  classpath 'com.neenbedankt.gradle.plugins:android-apt:1.8'
}
```

Dentro del archivo app/build.gradle, debemos añadir el plugin antes de añadir las dependencias de Parceler.

```
plugin : 'com.neenbedankt.android-apt'

dependencies {
  compile 'org.parceler:parceler-api:1.1.5'
  apt 'org.parceler:parceler:1.1.5'
}
```

Utilizando Parceler a nuestra clase

Incluiremos un constructor vacío, las variables de clase deben ser public y finalmente utilizaremos la anotación **@Parcel**. Bien esto nos ahorra mucho código y al mismo tiempo este se hace mucho más legible. Pero tiene un inconveniente, asume como entorno toda la clase User. Para evitar encapsular variables que no son necesaria utilizaremos la anotación **@Transient**

```
@Parcel
public class User {
    // class vars must be public
    public String name;
    public String lastName;
    public String job;
    @Transient
    public boolean hasCar;

    // empty constructor needed by the Parceler library
    public User() {
    }

     public User(String name, String lastName, String job,
boolean hasCar) {
        this.name = name;
        this.lastName = lastName;
        this.job = job;
        this.hasCar = hasCar;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getJob() {
        return job;
    }
```

```
    public void setJob(String job) {
        this.job = job;
    }

    public boolean isHasCar() {
        return hasCar;
    }

    public void setHasCar(boolean hasCar) {
        this.hasCar = hasCar;
    }
}
```

Para crear un parcelable utilizaremos **Parcels.wrap**:

```
User user = new User("Joan", "Miquel", "Android Developer", true);
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
intent.putExtra("user", Parcels.wrap(user));
```

Para recuperar el objeto user utilizaremos **Parcels.unwrap**:

```
user = Parcels.unwrap(getIntent().getParcelableExtra("user"));
```

Observaciones

Como podéis comprobar el uso de esta librería es muy útil y ademas muy fácil incluir en nuestros proyectos. Pero no todo es perfecto, como toda librería que utilicemos, mejorará con el tiempo y solucionara bugs. A priori esto es bueno pero no disponemos de un sistema de upgrade automático. Debemos de estar detrás de las novedades y preocuparnos notros de actualizarla. Tenemos que tener en cuenta que utilizar esta librería tiene sus ventajas e inconvenientes. Nos facilita mucho el uso de parcelabe pero al mismo tiempo es menos eficiente que implementarlo nosotros mismos. De echo el uso de esta librería es un buen balance entre eficiencia y facilidad de uso en comparación con una serialización y una

implementación de parcelabe. Cabe destacar que si nuestra aplicación requiere de la máxima eficiencia lo mejor será implementar parcelable. A continuación os dejo el enlace a la página de la librería, un simple ejemplo en GitHub y un artículo en android developers de como usar parcelables

- Parceler
- ParcelerTest
- Parcelable (Android Developer)

Recuperar Lg Optimus 3D (Unbrick, Unroot, UnRecovery)

Introducción

En este post voy a mostrar como recuperar un Lg Optimus 3D que no arranca. Lo primero es coger un poco de aire. Es importante (aunque sea difícil de conseguir) tranquilizarse un poco para seguir los pasos adecuadamente. Por suerte este dispositivo tiene un modo de recuperación que se ejecuta a muy bajo nivel,

así que es muy difíl (que no imposible) dar por perdido por completo el dispositivo. Al acabar el proceso, si todo ha ido adecuadamente, tendremos nuestro Lg funcionando sin acceso a root, con el recovery original y la versión más actualizada de la Rom ofrecida por Lg.

Consideraciones previas:

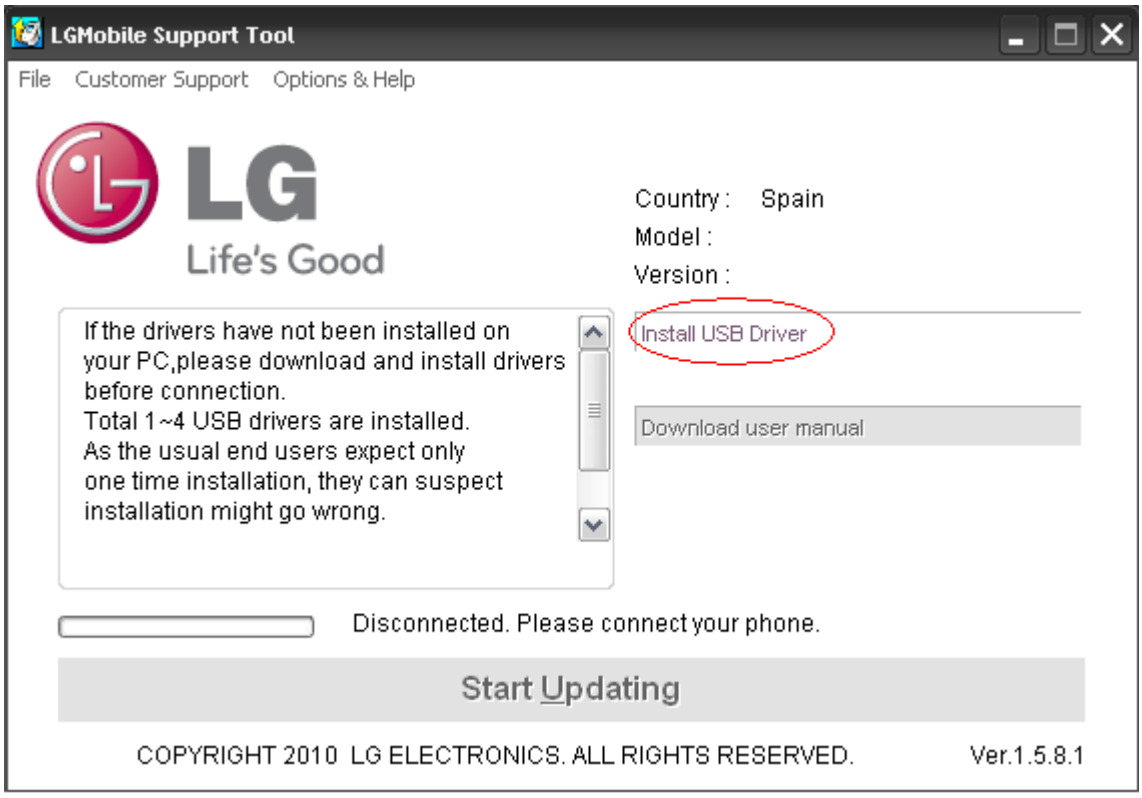
Bien en ningún caso me hago responsable de ningún daño ocasionado a algún terminal. Esta guía y las aplicaciones que se mencionan en ella carecen de cualquier tipo de garantía (un software de LG que decir…). Es decir, bajo vuestra responsabilidad queda lo que hagáis con vuestro terminal.

– Requisitos:

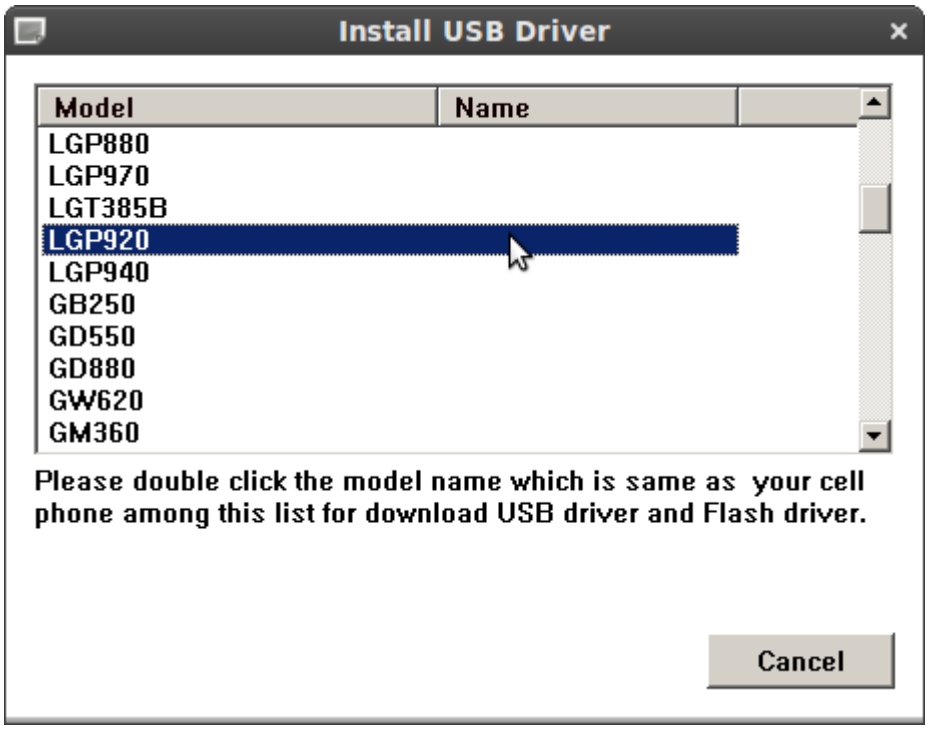
- Herramienta de recuperación oficial -> B2CAppSetup.
- Un pc con Windows. Si lo sé, pero que le haremos la herramienta de recuperación es para windows y el que se encuentre en esta situación no esta para experimentos con wine. También puntualizar que con màquina virtual no funciona (por lo menos con VirtualBox).
- El dispositivo debería tener suficiente batería (en torno un 50%), es un requerimiento difícil pero de lo contrario podríamos quedarnos a media instalación.
- Conexión a internet, la rom oficial que se instalará será bajada de internet.
- Necesitaremos el IMEI y el Numero de serie, podemos encontrarlos debajo de la batería.

Pasos a seguir

- Instalar drivers -> Arrancaremos B2CAppSetup y clicamos en install drivers.


LGMobile Support Tool
File
Customer Support
Options & Help
LG
Life's Good
Country : Spain
Model :
Version :
If the drivers have not been installed on your PC,please download and install drivers before connection.
Total 1~4 USB drivers are installed.
As the usual end users expect only one time installation, they can suspect installation might go wrong.
Install USB Driver
Download user manual
Disconnected. Please connect your phone.
Start Updating
COPYRIGHT 2010 LG ELECTRONICS. ALL RIGHTS RESERVED.
Ver.1.5.8.1

- Seguidamente seleccionamos **LGP-920** e instalamos el driver.

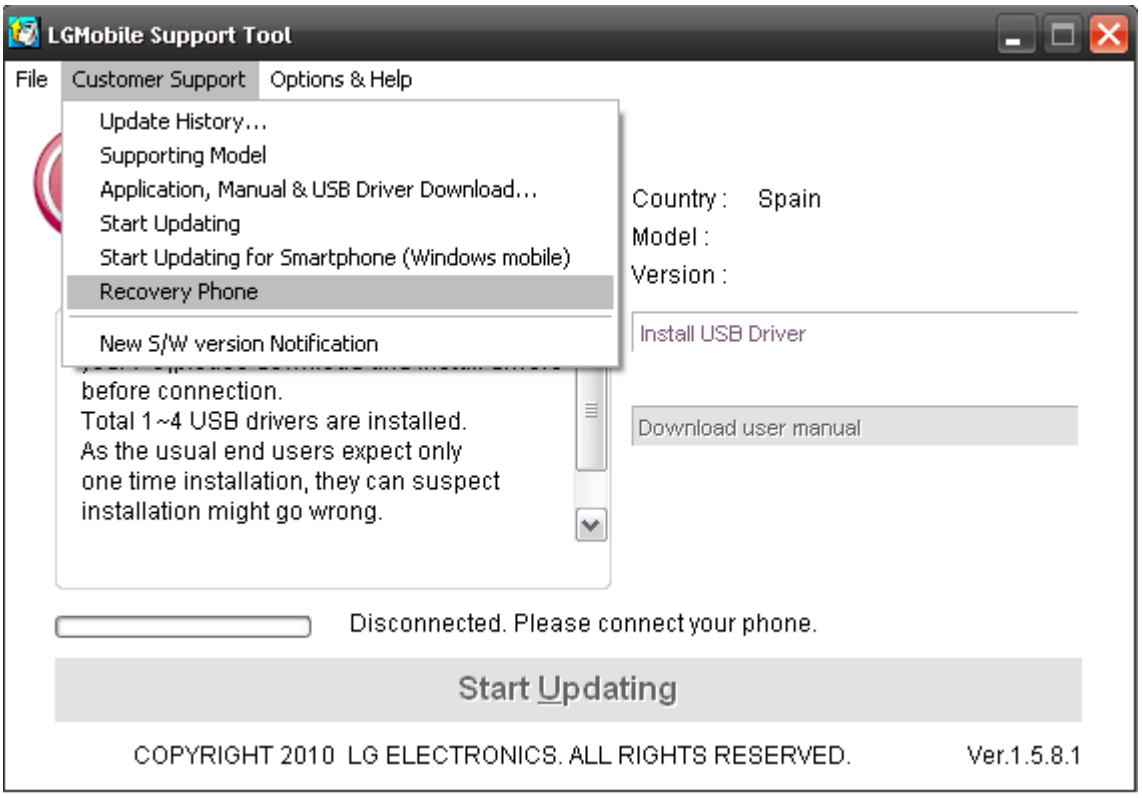


- Arrancar Lg Optimus 3D en modo recuperación

1. *Con el dispositivo desconectado del ordenador le quitamos la bateria y la volvemos a poner.*
2. *Introducimos la batería y conectamos el dispositivo al ordenador. E$l dispositivo arrancara saldrá el logo de LG seguidamente la batería y se apagará.*
3. *Presiona el botón de subir el volumen y el botón de encendido y no los sueltes hasta que salga el logo de LG y se vuelva apagar la pantalla. Con la pantalla apagada vuelve presionar los botones hasta que en la barra de notificaciones de windows veas que se empiezan a instalar los drivers. Puede ser que no te salga el símbolo de la batería, si a la primera no funciona tomatelo con calma y repite los pasos.*
4. *Una vez instalado el driver en el programa de recuperación de LG nos saldrá que el dispositivo está conectado.*

- Recuperación

    1. *Cuando el programa nos haya reconocido el dispositivo debemos selecionar Recovery Phone*



1. *Seguidamente una nueva ventana nos pedirá que introduzcamos el IMEI y el Numero de serie, que previamente hemos apuntado, una vez introducidos clica en el CHECK del IMEI y automáticamente comenzará el proceso de recuperación. Hay que tener en cuenta que se bajará la Rom de internet, esto quiere decir que dependiendo de tu connexión el proceso puede tardar más o menos (unos 30 min.), paciencia.*
2. *Una vez acabada la instalación desconnecta el dispositivo del ordenador, quita la batería, ponla, arranca y listo.*

Observaciones

Espero que esto pueda servir de ayuda. Normalmente esto ocurre cuando se flashean Roms de diferente baseband. Se debe tener cuidado siempre que flasheemos una Rom para evitar estos problemas. No nos veríamos en estas situaciónes (bricks producidos por instalar roms funcionales) si Lg tuviera un poco más en cuenta la calidad de su software…

Ruben.

Root, CyanogenMod 7, Ultra Smooth CyanogenMod 9.1 para HTC WildFire

Introducción

En este post voy a mostrar como trastear un HTC WildFire. Hace ya tiempo adquirí uno de estos terminales. Con amargura vi como se quedaba en la versión 2.2 y nunca se iba a actualizar a la versión 2.3.x de Android. Esto implicaría perder las siguientes mejoras:

- Función copiar y pegar mejorada
- Teclado rediseñado
- Gestor de descargas

- Manejo de energía mejorado y control de aplicaciones

Ciertamente la parte más importante es la del manejo de aplicaciones i gestión de la batería. Aunque la mejoras de usablidad siempre son bienvenidas, la verdad. También hay que tener en cuenta que estas no son todas las diferencias pero si las que se aplican a este terminal. Ademas esta rom nos permite personalizar muchos aspectos como transición entre pantallas, menús… Incluye AWD Launcher que no esta nada mal y podemos personalizarla con multitud de temas.

Después de mucho pensarlo decidí dar el paso rootear el terminal e instalar alguna rom que me permitiera disfrutar de la nuevas mejoras. El proceso fué un poco complicado e implica cierto riesgo, pero la mejora es incerible. La verdad después de probar CyanogenMod aunque la rom oficial se actualizase a 2.3 no cambiaría.

Consideraciones previas

Bien en ningún caso me hago responsable de ningún daño ocasionado a algun terminal. Esta guía y las aplicaciones que se mencionan en ella carecen de cualquier tipo de garantía. Es decir, bajo vuestra responsabilidad queda lo que hagáis con vuestro terminal.

Es recomendable que la batería del termina este cargada, realizar backups y asegurarse de que en la sd tenemos espacio para estos y las bajada que vamos a realizar. A continuación mostraremos algunas aplicaciones que nos pueden ser de ayuda:

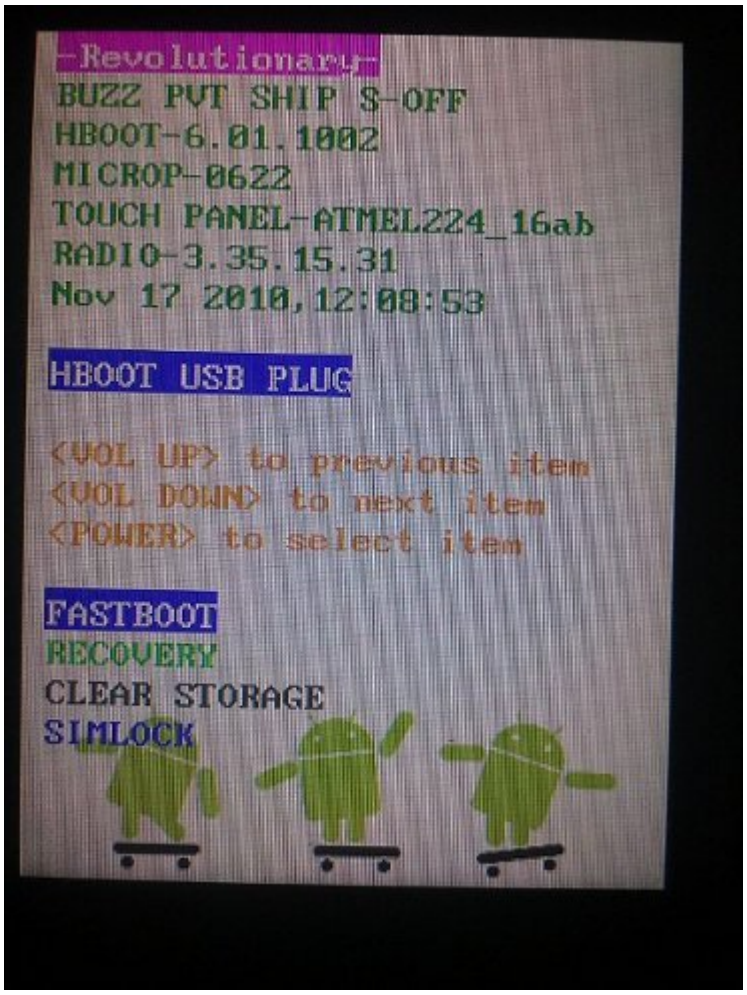
- Contactos del teléfono -> No requiere aplicación externa. Simplemente entrar en la lista de contactos, apretar el botón de menú de Android y seleccionar Importar/exportar -> Exporta a la tarjeta SD.
- Registro de llamadas -> Call Log Backup & Restore

- Registro de sms -> SMS Backup & Restore
- Aplicaciones -> MyBackup (ATENCION!! Esta es una versión de prueba funcional solo durante 30 días.)

S-OFF e instalación de un custom recovery

Esta parte es un poco complicada conviene seguir todos los pasos con paciencia y asegurándonos que realizamos cada uno correctamente.

- Debemos descargar Revolutionary.
- En cuanto empiece la descarga, la pagina se actualizará. Para poder utilizar el programa necesitamos una beta key. Para conseguirla es necesario rellenar el formulario que aparecerá introduciendo nuestra versión de HBOOT y número de serie.
    - HBOOT -> para encontrar la versión de HBOOT debemos apagar el terminal y encenderlo apretando el botón de volumen hacia abajo y el botón de encendido. Aparecerá una pantalla parecida a la siguiente en la que podemos ver la versión de HBOOT.



- El número de serie lo podemos encontrar debajo de la batería o bien, con el terminal conectado por usb, ejecutando en un terminal de pc

  ./adb devices

- Una vez realizados los pasos anteriores, debemos ejecutar Revolutionary

  ./revolutionary

  desde un terminal de pc, con nuestro WildFire conectado por usb, e introducir la beta key.

- Nuestro terminal se reiniciará en poco tiempo. El proceso puede no funcionar a la primera, repetidlo. En caso de tener algún problema podéis encontrar información en la wiki de Revolutionary o en su canal de irc.

Instalar CyanogenMod 7 (Android 2.3.7) y GoogleApps

Hace pocos días el Team de CyanogenMod sacó la versión 7.2 estable. Lo cual es una gran noticia. Las versiones estables son realmente buenas. Normalmente la segunda release candidate de una versión ya suele estar muy bien.

Bien podemos realizar este proceso desde Rom Manager:

- Arrancamos la aplicación y seleccionamos **Download Rom**.
- Seguidamente en el apartado **Free** seleccionamos **CyanogenMod**.
- Seleccionamos la última stable release, en este caso la **7.2.0**. y la opción de **GoogleApps**.
- Cuando finalicen las descargas debemos seleccionar **Wipe Data** y **Wipe Cache**, seria muy recomendable realizar un backup de la rom existente si queréis hacerlo seleccionad **Backup Existing ROM**.
- La aplicación SuperUser nos preguntará si queremos dar permisos a Rom Manager, aceptamos.
- Nuestro terminal se reiniciará en modo recovery hará un wipe de cache y data, a continuación se instalará CyanogenMod y las GoogleApps. Cuando finalice la instalación el terminal se reiniciará otra vez y CyanogenMod arrancará por primera vez.

O bien manualmente, es recomendable utilizar Rom Manager puesto que simplifica el proceso. En este caso debemos descargar:

- La rom cm-7.2.0-buzz
- Las GoogleApps gapps-gb-20110828-signed
- Una vez descargadas debemos copiarlas a la raiz de la targeta SD, entrar en Rom Manager y seleccionar **Reboot into Recovery**.

- Una vez en el recovery debemos utilizar las teclas de volumen para navegar y el botón de encendido o el trackball como enter.
- Bien si queréis realizar un backup de la rom actual (cosa recomendable porque es la original) entrad en **backup and restore** y seleccionad backup. Finalmente seleccionad **+++++Go Back+++++** para volver al menú principal.
- En el menú principal seleccionad **Wipe data/factory reset** y seguidamente **Wipe cache partition**.
- Seleccionad **Install zip from sdcard y Choose zip from sdcard**.
- Seleccionad **cm-7.2.0-buzz.zip** y esperad a que acabe el proceso de instalación.
- Bien para instalar las GoogleApps el proceso es el mismo que con la rom, simplemente debemos seleccionar **gapps-gb-20110828-signed**.
- Finalizada la instalación seleccionamos **+++++Go Back+++++** para ir al menú principal y finalmente **Reboot system now** option. Nuestro terminal se reiniciará y CyanogenMod arrancará por primera vez

Tips & tricks

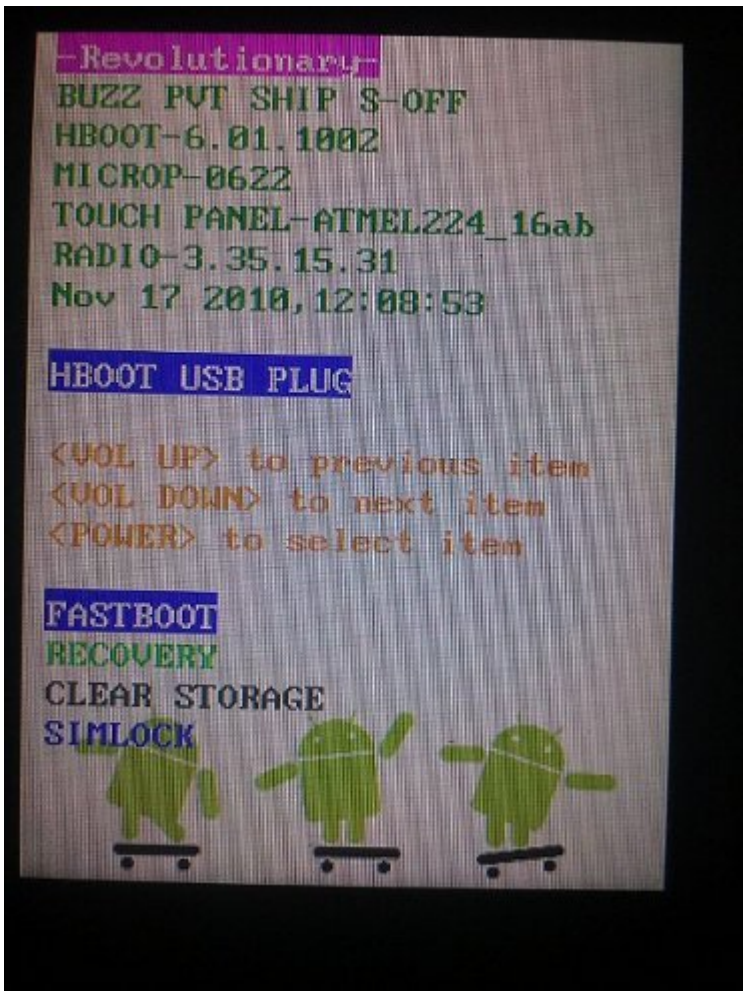
Hay algunas cosas a tener en cuenta:

- Ubuntu no reconoce el terminal. Debéis situaros en la carpeta donde se encuentra adb. Matar el servidor y arrancarlo como root

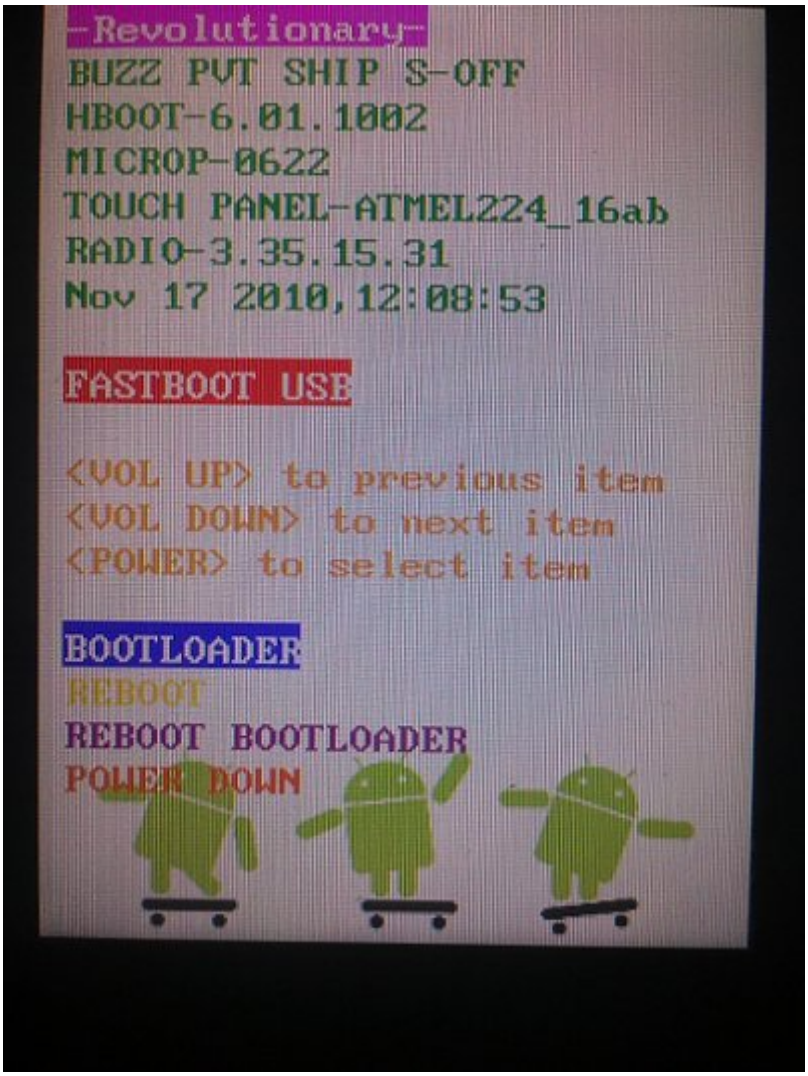
  ```
  ./adb kill-server
  sudo ./adb start-server
  ```

- Puede que el gps no coja señal. Eso es debido a que hay que flashear una nueva radio. Podéis descargar la radio des de aquí . Una vez descargada la radio debéis

descargar el siguiente ejecutable FastBoot. Debéis apagar el teléfono, y estando este conectado por usb, arrancar presionando el botón de encendido y volumen abajo. Entrareis en la siguiente pantalla.



- Con las teclas de volumen debéis seleccionar **FASTBOOT** y clicar el botón de encendido. Llegareis a esta nueva pantalla.



- Llegados a este punto des de un terminal en el pc debéis ejecutar

  ./fastboot flash radio radio.img

  Una vez acabe la instalación seleccionáis con los botones de menú **REBOOT** y presionáis el botón de encendido para reiniciar el terminal, con esto el gps debería funcionar perfectamente. Si aún así continuáis teniendo problemas podéis encontrar soporte en este foro CyanogenMod Forum.

- La aplicación de GoogleMpas se queda colgada. Desafortunadamente nuestro terminal no soporta la última versión de google mpas. Podéis descargar el apk des de aquí, es la versión 5.0.0 que funciona perfectamente.

- No se puede instalar flash. Podéis descargar una versión funcional de flash para este terminal des de aquí

Actualización a ICS

Recientemente he estado probando otra Rom, Ultra Smooth CyanogenMod 9.1.Tanto su estabilidad, eficiencia y compatibilidad con el hardware del terminal son impresionantes. La camara funciona bastante bien (excepto la gravación de video) y youtube (excepto el visionado en HD) . Algunos consejos:

– En opciones -> Performance -> Processor podéis modificar la velocidad del procesador entre 352 y 710 MHz y el governor SMARTASSV2 (estaremos realizando overclocking esto comporta cierto riesgo pero dentro de unos valores razonables).

– Podéis modificar en las opciones del launcher (menu de android -> launcher settings – Desktop settings) Wallpaper scrolling para desactivarlo.

– Podéis encontrar unas tiny google apps aquí simplemente una vez cargada la página de Google Docs persionad Ctrl+s o en el menu File->download

– En este post podéis encontrar como instalar Google VoiceSearch

– Es recomendable utilizar Seeder(consigue reducir el Lag notablemente)

Observaciones
Bien, el proceso es un poco complicado y algunos pasos son críticos, pero a mi me mereció la pena el cambio. Ahora estoy muy contento con mi terminal, hay muchas opciones que la rom oficial no ofrece, la estabilidad es muy buena así como

también el ahorro de barrería y la gestión de memoria (cuando htc sence desaparece, de las entrañas de nuestro terminal memoria ram aflora). También cabe tener en cuenta el echo de ser root. Se pueden realizar backups de todo, hay muchas aplicaciones que explotan esta funcionalidad, pero esta parte os la dejo para vosotros.
En la nueva parte para instalar ICS podemos observar varios ajustes, seamos realista estamos hablando de correr una ICS 4.0.4 en un HTC WildFire sorprendente es que funcione y lo bien que lo hace la Rom Ultra Smooth CyanogenMod 9.1, en la fuentes se encuentra el link al post de xda del creador y colaboradores de la rom.

Fuentes

- CyanogenMod
- Ultra Smooth CyanogenMod 9.1

Ruben.

Montar BlackBerry PlayBook en Ubuntu

Introducción

En este post voy a mostrar como montar una BlackBerry PlayBook a Ubuntu.Algo que a priori debería ser sencillo y lo es con un perqueño workaround.

Cuando se conecta la PlayBook mediante cable usb a Ubuntu una pantalla nos indica que se están instalando los drivers… eso no va a ocurrir en Ubuntu, simplemente cerramos esa pantalla.

Bien la idea consiste en conectarla por red, los pasos a seguir son realmente sencillos.

Preparar la PlayBook

- Iremos a Configuración -> Almacenamiento y uso compartido. En conexión usb seleccionaremos Conectarse a Windows.
- Seguidamente iremos (dentro de Configuración) a -> Sobre (la sección donde encontramos la información relacionada con el dispositivo). En Visualizar Información de la tablet seleccionaremos Red. Deberemos apuntarnos la dirección IP de la PlayBook para usarla después.

Conectarnos con Ubuntu

- Iremos a Lugares -> Conectarnos a un Servidor.
- En el formulario de conexión seleccionaremos Recurso Compartido de Windows e introduciremos la IP que anteriormente hemos apuntado y con esto accederemos mediante red al almacenamiento masivo de la PlayBooK.

Observaciones
Bien, esto no es más que un truquillo simple pero a su vez muy útil. Próximamente mostraré como portar una aplicación Android a PlayBook y las limitaciones que existen.

Ruben.

DESBLOQUEAR BLACKBERRYS 88XX – 99XX

COMO DESBLOQUEAR (LIBERAR) TU BLACKBERRY 88xx y 99xx GRATIS

Programas que hay que descargar:
1. Microsoft Net Framework 2.0 (solo si no lo tenemos) http://go.microsoft.com/fwlink/?linkid=32168
2. Desktop Manager https://www.blackberry.com/Downloads/entry.do?code=A8BAA56554F96369AB93E4F3BB068C22
3. Sistema Operativo compatible tu modelo BlackBerry http://na.blackberry.com/eng/support/downloads/download_sites.jsp (busca el país, tu operador y después el modelo de tu BB)
4. Borrar el archivo VENDOR.XML (C:\Program Files\Common Files\Research In Motion\AppLoader) (cada vez que uses el programa es aconsejable que uses este punto para liberar otra, ya que vuelve a aparecer cada vez)
5. Descargar e Instalar el MFI Multiloader : http://rapidshare.com/files/182971340/Lista_links (te abrira un texto, y bajas los 4 archivos de rapidshare), ejecuta el 5 archivo y te creara una carpeta MML. Unes los otros cuatro archivos y te crea otra carpeta MML, la cual la copias y la pegas encima de la creada primero, copiando todos los archivos.

PASOS A REALIZAR
Desconectamos internet, y ya tenemos instalados todos los programas anteriores.

1. Aun no conectamos la BB, abrimos MML/bin/mml (research in motion), y nos abre el programa que desbloquea la BB,

entonces enchufamos la BB, y el programa empieza a trabajar y pone BUSY, lo dejamos que ejecute los archivos hasta que suene el USB y ponga en verde PASS, a la vez nos saldrá en la pantalla de la BB UN “1”, lo cual quiere decir que ya está desbloqueada.

2. Lo siguiente ya es fácil, abres el desktop y te vas a application loader, y cargas el software de tu operador y de tu modelo, y te detectara tu bb, le das a aceptar y si lo quieres en español en una de las opciones marcas compatibilidad con idioma español, sino te saldrá en inglés y es un poco “coñazo”, estate atento por que después de borrar lo que haya en la BB y vaya a empezar a cargar suena el sonido del USB, y te sale una pantalla blanca en la BB, en la que tienes que apretar START, y continuará la carga, si te despistas, tendrás que volver a hacer este paso, por eso estate atento.
NOTA: El proceso tarda en total unos 20 minutos, ten carga para que te aguante ese rato.

Referencia:
http://www.taringa.net/posts/celulares/2263063/Liberar-Black-berry-manual-facil.html