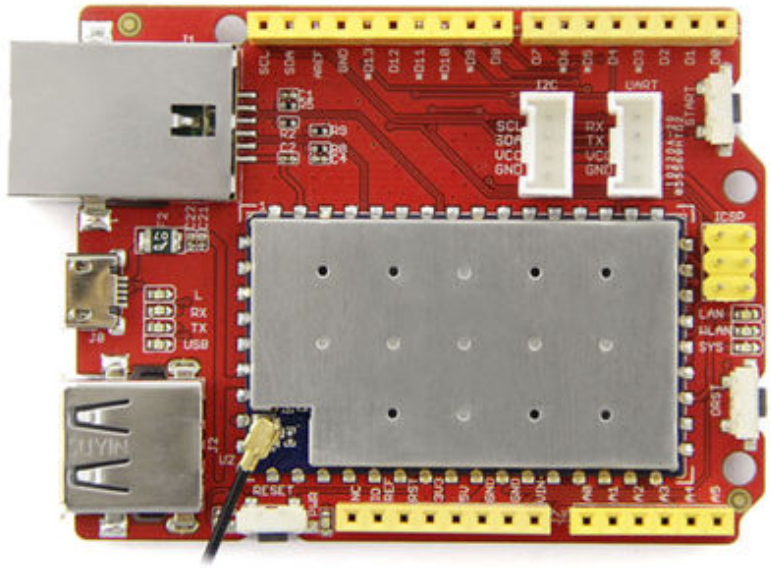
Seeeduino Cloud – Parte 2



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.

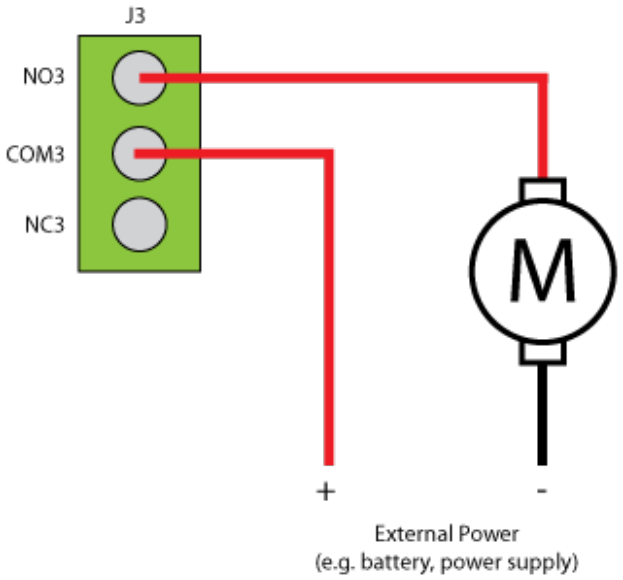
- Montando el Relay Shield y cableado de un relay
- Programa para controlar el relay
- Encendiendo y apagando nuestro ventilador

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.

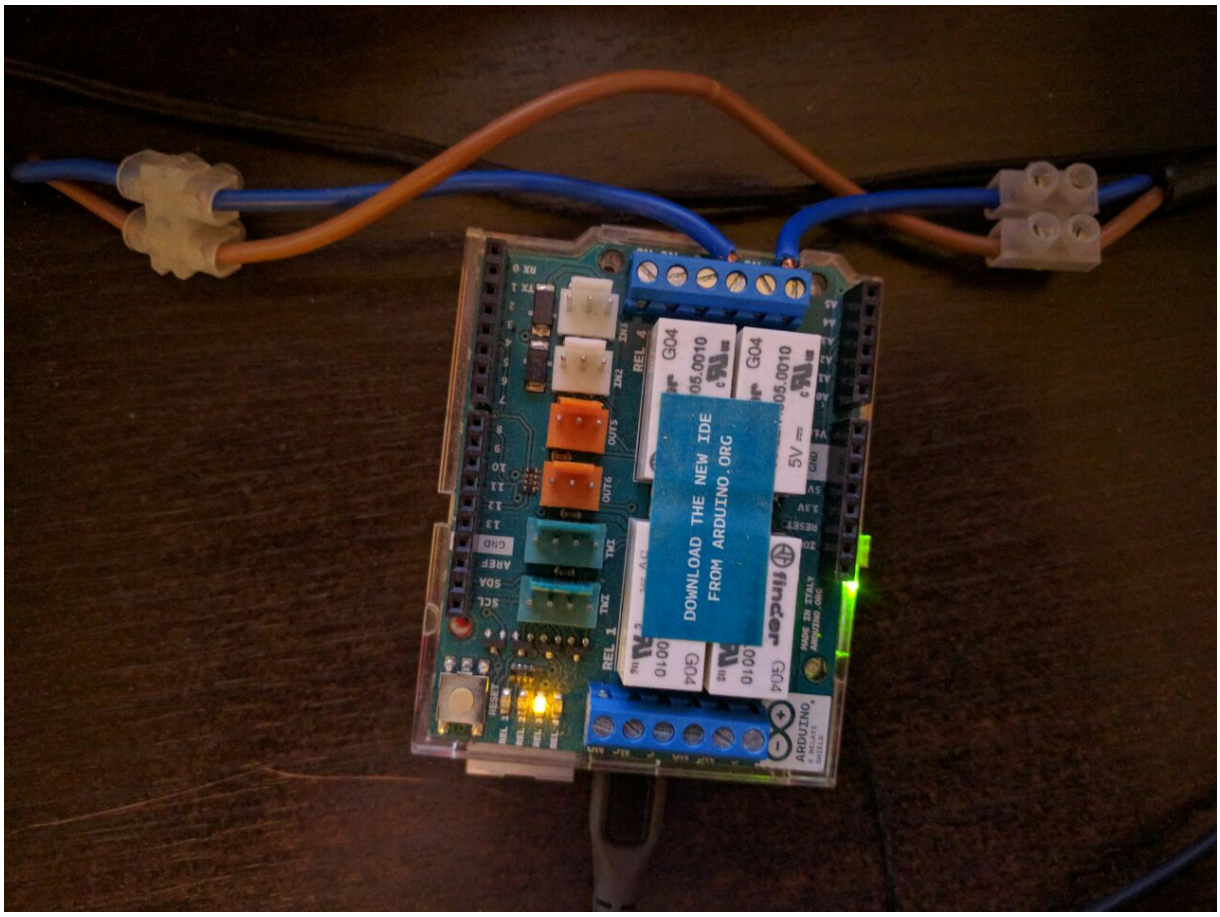


Motor-shield-schematic-drawing

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}

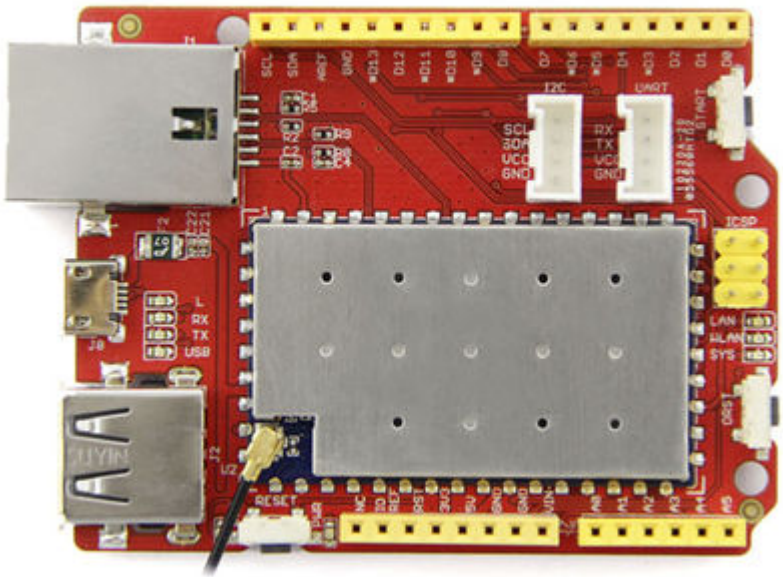
Conclusión

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

Seeeduino Cloud – Parte 1



Introducción

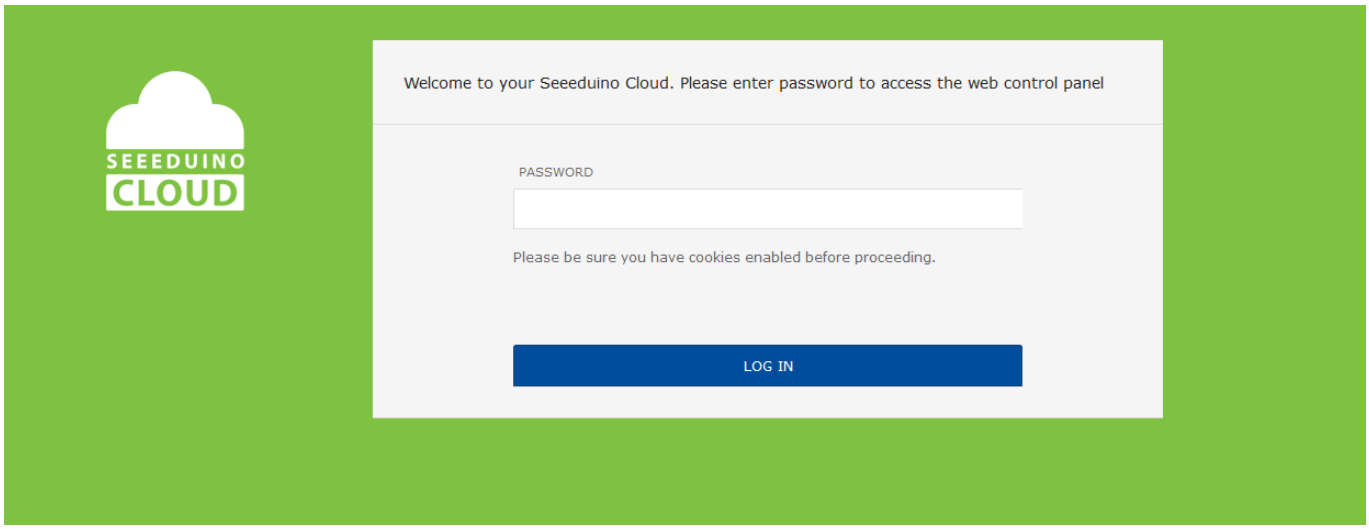
Recientemente he adquirido un Seeeduino Cloud. Un clon

compatible con Yun. La verdad es que es muy versátil, fácil de configurar y potente. En este primer post hablaremos un poco de él, lo configuraremos y habilitaremos su Api Rest para encender y apagar el led que incorpora la placa.

- Configuración de red
- Configuración con WebGui
- Programa de ejemplo y test

Configuración de red

Cuando arranquemos por primera vez nuestro Seeeduino, este levantará una wifi llamada **SeeeduinoCloud-AXXXX** a la que nos podremos conectar. Una vez echo esto, podremos acceder a la configuración web a través de la ip **192.168.240.1.**



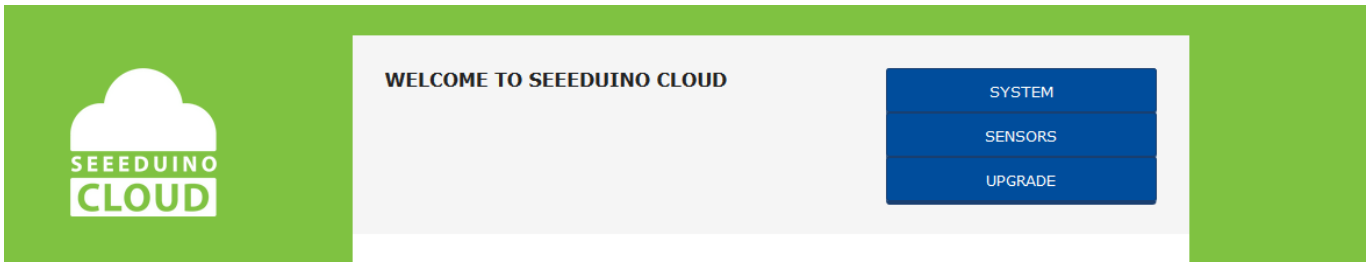
El password por defecto es **seeeduino**.

Configuración con WebGui

Después de entrar en la web, la interfaz nos mostrará el estatus de las redes WiFi/ Eth. En la parte superior derecha encontraremos las siguientes opciones:

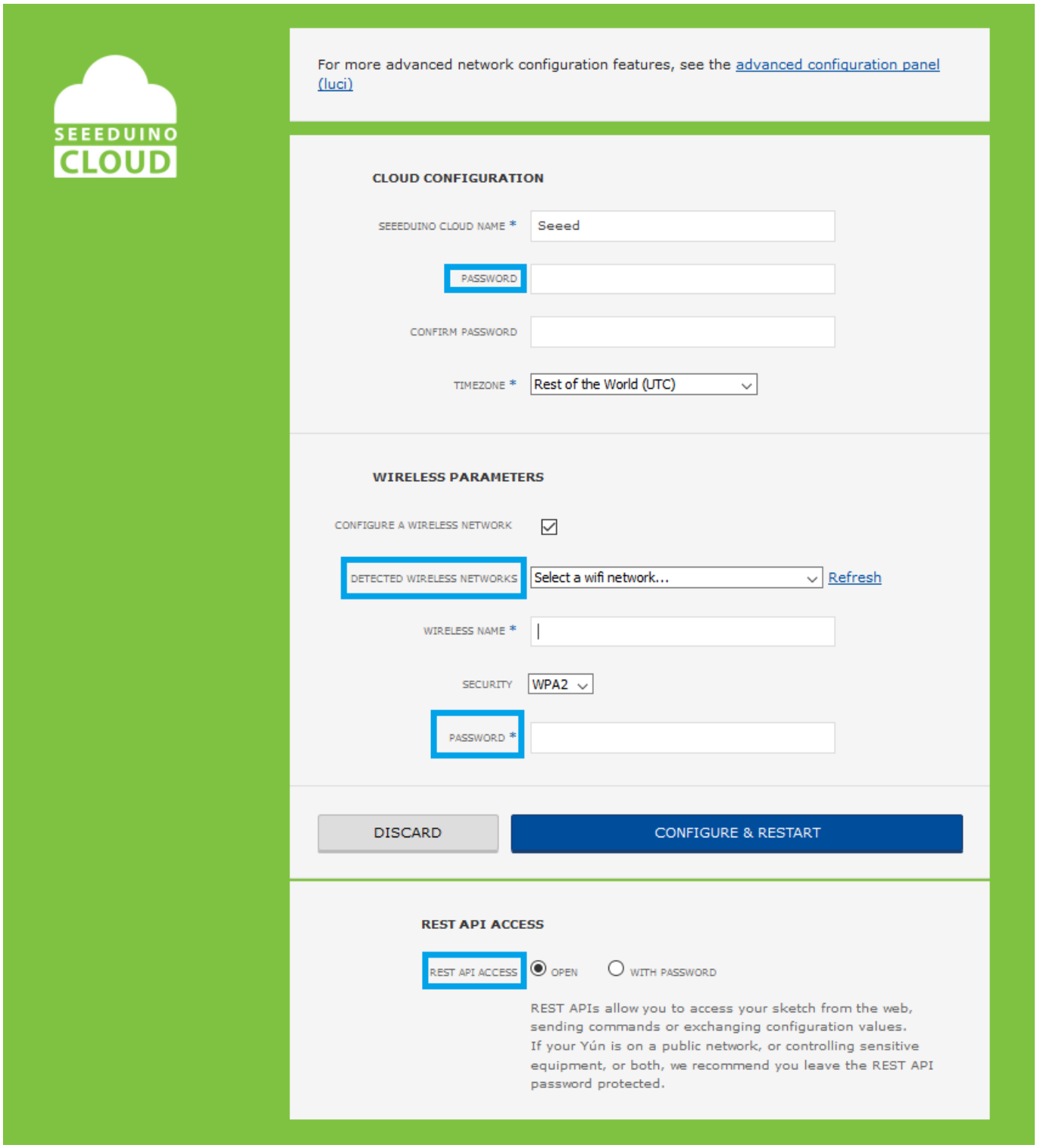
- SYSTEM -> Configuración global
- SENSORS -> configuración del servidor IoT

- UPGRADE -> Actualización de firmware



Seleccionaremos SYSTEM y configuraremos nuestro Seeeduino Cloud:

- Primeramente podemos modificar el password de acceso
- A continuación seleccionaremos la red wifi a la que queremos conectar nuestro Seeeduino Cloud y estableceremos el password de la misma
- Finalmente, podemos proteger nuestra red con una password para la Api Rest. Si lo hacemos debemos tener en cuenta que será una protección Basic Atuh y el usuario por defecto será root

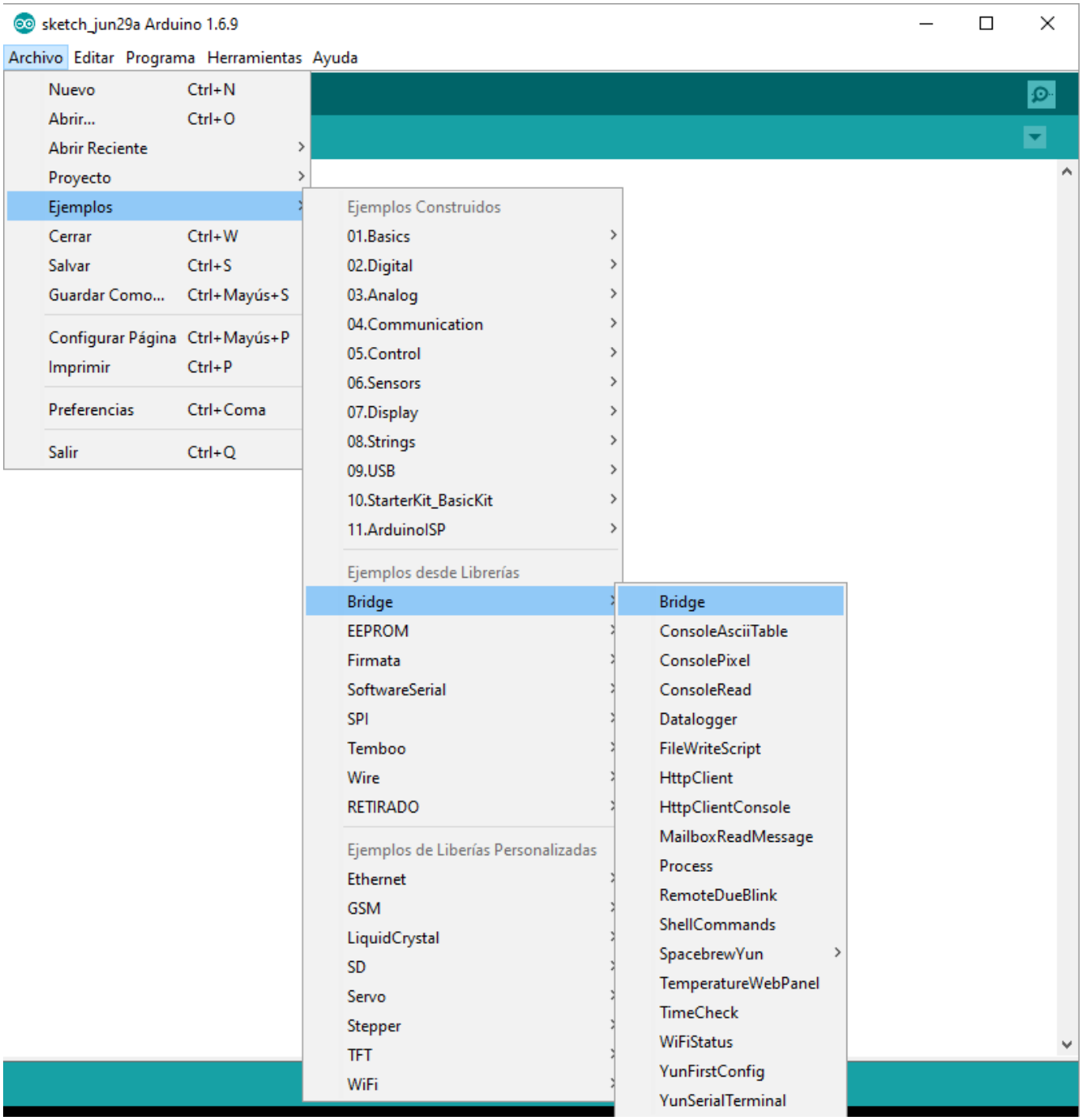


Cuando pulsemos en **CONFIGURE & RESTART** nuestro Seeeduino se configurará, se reiniciará y se conectará a nuestra red.

Programa de ejemplo y test

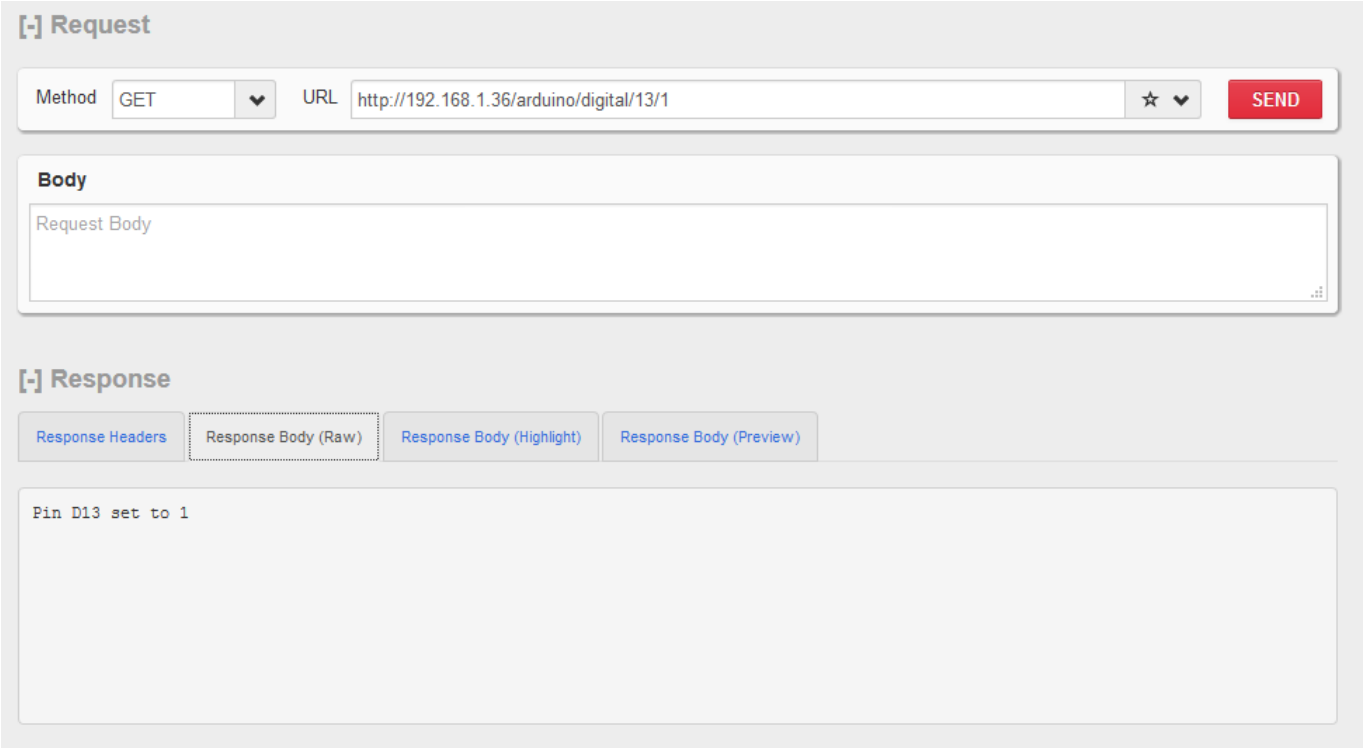
Bien lo siguiente sera subir un código de ejemplo al Arduino para poder testear la aplicación. Para más información sobre

como subir un programa des del IDE a nuestro Arduino podéis mirar este post Arduino – Primera Parte. Utilizaremos un ejemplo de la librería **Bridge.**



Una vez subido nuestro programa atacaremos la Api Rest para encender y apagar el led que vine incorporado en la placa, exactamente el 13 (situado justo detrás del puerto microusb). En esta caso utilizo el plugin para Firefox RESTclient, aunque podríamos utilizar cualquier otro. En la url especificamos la ip de nuestro Seeeduino, arduino(podemos acceder a otros

elementos de la placa), el tipo de pin (en este caso digital), el pin (13) y el valor (0 apagado, 1 encendido). La url final sería **http://192.168.1.36/arduino/digital/13/1** (encendido) o **http://192.168.1.36/arduino/digital/13/0** (apagado)



Conclusión

Como podemos ver la potencia de Seeeduino Cloud es mucha. Fácilmente podemos acceder a nuestro Arduino recibiendo y enviando valores a través de una sencilla Rest Api. En el próximo post acoplaremos un Relay Shield, esto nos permitirá controlar remotamente aparatos electrónicos (dado el calor que hace ahora, un pequeño ventilador)

Tutorial Arduino parte 4

Hola a todos .

Este es el cuarto tutorial sobre Arduino , en este tutorial vamos a continuar el tutorial 3 , creando una aplicación 3d para Android , que encienda la bombilla de forma remota .

Creamos un proyecto Android nuevo en eclipse con Android SDK.

Continuando como teníamos en el tutorial 3 , en la parte del servidor php , el fichero que recibía una variable GET y encendía o apagaba la bombilla si la variable es high o low.

En este primer código podemos ver como desde el proyecto Android podemos realizar la conexión con el servidor PHP para que envíe la señal a la placa Arduino.

El fichero PHP ha de estar en modo servicio , con las lineas //die… descomentadas , para que nos devuelva una respuesta.

```
private void lighting()
{
ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
if (cm != null) {
boolean connected=false;

NetworkInfo[] info = cm.getAllNetworkInfo();
if (info != null) {
for (int i = 0; i < info.length; i++) { if (info[i].getState() == NetworkInfo.State.CONNECTED) { connected = true; } } } if(connected) { String destiny="http://192.168.0.131/arduino/service.php?check=on"; String missatxe="Signal CHECK sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); switch(estado) { case 0: destiny ="http://192.168.0.131/arduino/service.php?hight=on"; missatxe="Signal ON sended. Click again to switch OFF";
```

```
cambiarTextoBoton("Switch OFF"); break; case 1: destiny ="http://192.168.0.131/arduino/service.php?low=on"; missatxe="Signal OFF sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); break; } escribir(missatxe); InputStream is = null; //initialize String result = ""; try{ HttpClient httpclient = new DefaultHttpClient(); HttpPost httppost = new HttpPost(destiny); HttpResponse response = httpclient.execute(httppost); HttpEntity entity = response.getEntity(); is = entity.getContent(); //convert response to string try{ BufferedReader reader = new BufferedReader(new InputStreamReader(is,"UTF-8"),8); StringBuilder sb = new StringBuilder(); String line = null; while ((line = reader.readLine()) != null) { sb.append(line + "\n"); } is.close(); result=sb.toString(); ArrayList message=this.parse( result);
String status=message.get(0).toString();
String signal=message.get(1).toString();
String mensage=message.get(2).toString();
if(signal.compareTo("1")==0)
{
checkOK=1;
estado=0;
}
else
{
if(signal.compareTo("2")==0)
{
checkOK=2;
estado=1;
}
else
{
if(signal.compareTo("-1")==0)
{
checkOK=-1;
estado=-1;
}
```

```
}

}
}catch(Exception e){
Log.e("log_tag", "Error converting result "+e.toString());
escribir("Error converting result "+e.toString());
}
}catch(Exception e){
Log.e("log_tag", "Error in http connection "+e.toString());
escribir("Error in http connection "+e.toString());
}
}
}

}
```

Bien , ya podemos comunicarnos con el servidor , pero bueno un poco rancia la aplicación , no?

¿Falta un botón , un poco de color o algo no?

Puesto a investigar un poco pués encontré una librería que permitía crear aplicaciones OPENGL de manera un poco más sencilla de lo normal , con un lenguage más para un principiante como yo que ir directamente a OPENGL de Android , que es un poco más rudo para alguien que no lo usa muy a menudo.

Esa librería es Min3D.

Min3D nos permite crear escenas 3D e importar objetos 3DS o OBJ , así como texturas y objetos básicos como esferas y cubos.

Tras descargar min3d de su repositorio , incluimos la carpeta al proyecto y podemos ver y probar los ejemplos.

Partiendo del ejemplo ExampleLoadObjFileMultiple.java , donde podemos ver como cargar un objeto desde un fichero .OBJ .

En este fichero podemos ver que hereda de la clase extends RendererActivity , y lo aplicaremos a la clase que estemos utilizando .

El método initScene() es el encargado de crear la escena , en este método es donde colocaremos todos los objetos de la escena , cargaremos todos los ficheros necesarios , .OBJ , .PNG , y crearemos las esferas y rectángulos que necesitemos.

Este método es llamado al iniciar la aplicación y cada vez que el dispositivo entra en modo PAUSE , o se bloquea la pantalla.

Como en toda aplicación 3D , además de una escena necesitamos una cámara y luces para iluminar nuestros objetos.

En este método asignaremos un color de fondo , o una textura , situaremos las luces , los objetos y situaremos y enfocaremos la cámara , para obtener la perspectiva adecuada.

El método updateScene() se ejecuta en cada frame , sería el equivalente al típico evento draw() de repintado de la pantalla en aplicaciones 2D , osea que se ejecuta cada vez que se pinta la pantalla.

Éste método es el encargado de las instrucciones de las animaciones , por ejemplo si queremos rotar un objeto , en este método calculamos el valor nuevo de la rotación y se le asigna.

Para cargar los ficheros .OBJ en la aplicación Android es necesario modificar los nombres de los archivos.

Los ficheros .OBJ suelen ir acompañados de un fichero con el mismo nombre con extensión .mtl con la definición de los materiales de los grupos de los objetos que hayan definidos en el fichero .OBJ , además de las imágenes de las texturas.

Osea que el fichero .OBJ tiene objetos , valores de vértices y objetos y el fichero .mtl los materiales.

Las imágenes de las texturas las colocaremos en la carpeta res/drawable… .

Los ficheros .OBJ y .MTL los renombramos sustituyendo el punto por un «_» osea un guión bajo , para evitar los conocidos conflictos de ficheros con el mismo nombre y diferente extensión que tiene la programación con Android.

Para este tutorial hemos creado una farola , con una esfera que hace de bombilla , y una esfera con una textura con transparencia .PNG , que hará el efecto de que la farola está encendida.

También se ha creado una especie de bullofa que emite unas esfera con transparencia , emulando el envío de ondas , dando a entender que se está comunicando con el servidor , una esfera nos indicará según la textura que tenga el texto ON de color verde , o el texto OFF de color roja o de color ambar si no hay conexión con el servidor ,a además hay otra bola que nos indicará según su textura , si estamos a más de cierta distancia de la bombilla , si hay cobertura GPS .

Bueno abrimos blender y creamos una lámpara y una bullofa (algo que de el efecto de ser un emisor de ondas) , y las exportamos en formato wavefront .OBJ .

Si no queremos utilizar blender o otro software para crear objetos 3D , los podemos descargar de alguna página , por ejemplo ésta , que ofrezcan objetos 3D en formato .OBJ .

Podemos ver como asignar el color de fondo:

```
scene.backgroundColor().setAll(0xfff2d533);
```

Asignar una textura a un rectángulo:

```
Bitmap b = Utils.makeBitmapFromResourceId(this, R.drawable.scuraki);
float w = 20f;
float h = w * (float)b.getHeight() / (float)b.getWidth();
Rectangle suelo = new Rectangle(w, h, 1,1, new Color4());
```

```
suelo.doubleSidedEnabled(true); // ... so that the back of the
plane is visible
suelo.normalsEnabled(false);
scene.addChild(suelo);

Shared.textureManager().addTextureId(b, "scuraki", false);
suelo.textures().addById("scuraki");

Asignar una textura a una esfera:
b = Utils.makeBitmapFromResourceId(R.drawable.bolaroja);
Shared.textureManager().addTextureId(b, "bolaroja", false);
bolarojaTexture = new TextureVo("bolaroja");
esfera.textures().addReplace(bolarojaTexture);

Iluminar la escena:
luzobj = new Light();
luzobj.ambient.setAll(new Color4 (128,128,128,128));
luzobj.diffuse.setAll(new Color4 (64,64,164, 128));
luzobj.emissive.setAll(new Color4 (0,0,0,255));
luzobj.specular.setAll(new Color4 (0,0,0,255));
luzobj.type(LightType.POSITIONAL);
scene.lights().add(luzobj);
luzobj.position.setAll(0.65f, -0.85f, 3.5f);

Añadir un objeto .OBJ a la escena:
parser2 = Parser.createParser(Parser.Type.OBJ, getResources(),
"adictosalainformatica.min3DAdictos:raw/fanal_obj", true);

parser2.parse();

fanal = parser2.getParsedObject();
fanal.scale().y = 0.25f;
fanal.scale().z = 0.25f;
fanal.scale().x = 0.25f;
fanal.shadeModel(ShadeModel.SMOOTH);
fanal.vertexColorsEnabled(true);
fanal.normalsEnabled(true);
fanal.colorMaterialEnabled(false);
```

```
scene.addChild(fanal);
fanal.position().x=0.0f;
fanal.position().y=1.6f;
fanal.position().z=-5f;
fanal.rotation().x=25f;
```

Controlar la rotación de los objetos en el método updateScene:

```
bombilla.rotation().y=yrot;
bombilla.rotation().x=xrot;
receptor.rotation().y=yrot;
receptor.rotation().x=xrot;

xrot += xspeed;
yrot += yspeed;
_count++;
```

Para dar un toque de color la bullofa cambia el valor de escalado cada cierto tiempo , y se crean unas esferas con transparencia , que viajan desde la bullofa (que está cerca de la cámara) hasta la lámpara (que está al fondo de la escena), emulando el envio de ondas.

Un array de esferas ameniza la pantalla , desplazandose .

La clase Burbuja es la encargada de controlar estas esferas:

```
package adictosalainformatica.min3DAdictos;

import android.graphics.Bitmap;
import min3d.Shared;
import min3d.Utils;
import min3d.core.Scene;
import min3d.objectPrimitives.Sphere;
import min3d.vos.Color4;
import min3d.vos.TextureVo;
public class Burbuja {
private float velocidad_x=.000f;
private float velocidad_y=0.02f;
private float velocidad_z=0.04f;
private float posicion_x=0.45f;
```

```
private float posicion_y=-0.25f;
private float posicion_z=0.3f;
private float variacion_x=0.45f;
private float variacion_y=-0.25f;
private float variacion_z=0.6f;
private float limite_x=4.01f;
private float limite_y=4.01f;
private float limite_z=4.8f;
private Color4 color=null;
private Sphere esfera =null;
long _index;
private float contador=0;
public Burbuja(long index)
{
this._index=index;
}
public void make(Scene scene )
{
if(esfera!=null)
{
esfera.clear();
}
esfera=null;
TextureVo textura=null;
esfera = new Sphere(1.5f, 20,20);
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f;
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
Bitmap                              b                              =
Utils.makeBitmapFromResourceId(R.drawable.tscuraki);
Shared.textureManager().addTextureId(b, "burbuja0" + _index,
false);
b.recycle();
textura = new TextureVo("burbuja0" + _index);
esfera.textures().addReplace(textura);
esfera.colorMaterialEnabled(false);
esfera.vertexColorsEnabled(false);
esfera.lightingEnabled();
```

```
scene.addChild(esfera);

//Log.v(Min3d.TAG, "ReCrea Burbuja=" + _index);

}

public int mover()
{
int moviendo=1;
posicion_x=variacion_x + ( contador * velocidad_x);
posicion_y=variacion_y + ( contador * velocidad_y);
posicion_z=variacion_z - ( contador * velocidad_z);
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
if((posicion_x>limite_x)||(posicion_y>limite_y)||(posicion_z *
-1 > limite_z ))
{
esfera.clear();

moviendo=0;
}
//Log.v(Min3d.TAG, "Mueve Burbuja=" + _index + " posiciónx="+
posicion_x + " posicióny=" + posicion_y + " posiciónz=" +
posicion_z);
esfera.rotation().x=contador * -1.3f ;
esfera.rotation().y=contador * -1.3f;
esfera.rotation().z=contador * -1.3f;
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f +
(contador * 0.001f);
contador++;
return moviendo;

}

public float getLimite_x() {
return limite_x;
}
public void setLimite_x(float limite_x) {
this.limite_x = limite_x;
}
```

```
public float getLimite_y() {
return limite_y;
}
public void setLimite_y(float limite_y) {
this.limite_y = limite_y;
}
public float getLimite_z() {
return limite_z;
}
public void setLimite_z(float limite_z) {
this.limite_z = limite_z;
}
public Color4 getColor() {
return color;
}
public void setColor(Color4 color) {
this.color = color;
}
public float getVelocidad_x() {
return velocidad_x;
}
public void setVelocidad_x(float velocidad_x) {
this.velocidad_x = velocidad_x;
}
public float getVelocidad_y() {
return velocidad_y;
}
public void setVelocidad_y(float velocidad_y) {
this.velocidad_y = velocidad_y;
}
public float getVelocidad_z() {
return velocidad_z;
}
public void setVelocidad_z(float velocidad_z) {
this.velocidad_z = velocidad_z;
}
public float getPosicion_x() {
```

```
return posicion_x;
}
public void setPosicion_x(float posicion_x) {
this.posicion_x = posicion_x;
}
public float getPosicion_y() {
return posicion_y;
}
public void setPosicion_y(float posicion_y) {
this.posicion_y = posicion_y;
}
public float getPosicion_z() {
return posicion_z;
}
public void setPosicion_z(float posicion_z) {
this.posicion_z = posicion_z;
}
public float getVariacion_x() {
return variacion_x;
}
public void setVariacion_x(float variacion_x) {
this.variacion_x = variacion_x;
}
public float getVariacion_y() {
return variacion_y;
}
public void setVariacion_y(float variacion_y) {
this.variacion_y = variacion_y;
}
public float getVariacion_z() {
return variacion_z;
}
public void setVariacion_z(float variacion_z) {
this.variacion_z = variacion_z;
}
public Sphere getEsfera() {
return esfera;
```

```
}
public void setEsfera(Sphere esfera) {
this.esfera = esfera;
}
public long getIndex() {
return _index;
}
public void setIndex(long _index) {
this._index = _index;
}

}
```

Finalmente controlamos la distancia a la bombilla gracias al sensor GPS , activándolo si es necesario.

Podemos decidir activar la bombilla a cierta distancia , útil para luces de garages , por ejemplo que se encienda cuando falta 2 kilómetros para llegar con el coche.

En nuestro caso tenemos la esfera que nos indica la distancia , modificamos la textura para que nos muestre FAR si está lejos y NEAR si está cerca.

```
private void loadJipiEs()
{
// Acquire a reference to the system Location Manager
this.locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

if
(!this.locationManager.isProviderEnabled(LocationManager.GPS_P
ROVIDER))
{
createGpsDisabledAlert();
}
else
{
```

```
// List all providers:
List providers = this.locationManager.getAllProviders();

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);

this.bestProvider                                    =
this.locationManager.getBestProvider(criteria, false);

Location                  mylocation                 =
this.locationManager.getLastKnownLocation(this.bestProvider);

if(mylocation!=null)
{
this.getLocation(mylocation);
}
else
{
//this.afegir("Posición inicial vacía.");
}
}

}
private void getLocation(Location location)
{
if(location!=null)
{

boolean hasAltitude=false;
boolean hasAccuracy=false;
boolean hasBearing=false;
lat=location.getLatitude();
lon=location.getLongitude();
alt=location.getAltitude();

hasAltitude=location.hasAltitude();
hasAccuracy=location.hasAccuracy();
hasBearing=location.hasBearing();
```

```
distance=location.distanceTo(coords);
if(distance<5000) { distanceState="near"; } else { distanceState="far"; } } else { distanceState="Unknown"; }
//Toast.makeText(getApplicationContext(), "Distance is : " + distance, Toast.LENGTH_LONG).show(); /*runOnUiThread(new Runnable() { public void run() { Toast.makeText(getApplicationContext(), "Distance is : " + distance, Toast.LENGTH_LONG).show(); missatger.setText("Distance is : " + distance); } });*/
escribir("Distance is : " + distance); } private void createGpsDisabledAlert(){ AlertDialog.Builder builder = new AlertDialog.Builder(this); builder.setMessage("Your GPS is disabled! Would you like to enable it?") .setCancelable(false) .setPositiveButton("Enable GPS", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ showGpsOptions(); } }); builder.setNegativeButton("Do nothing", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ dialog.cancel(); } }); AlertDialog alert = builder.create(); alert.show(); }
private void showGpsOptions(){ Intent gpsOptionsIntent = new Intent(
android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
startActivity(gpsOptionsIntent); } }
```

En nuestro caso la luz la hemos activado a través de un botón externo a la escena , esta señal modifica la textura de la esfera que nos indica el estado de la bombilla , según sea la respuesta del servidor , ON , OFF o sin conexión , si la bombila está ON la esfera que emula la bombilla encendida , que está en la lámpara , se hace visible rodeando una esfera más pequeña que hace de núcleo de la bombilla.

Tutorial Arduino parte 3

En este tutorial veremos como conectar la placa Arduino a la red de 220V de nuestra casa , y poder utilizarlo para controlar nuestros aparatos eléctricos .

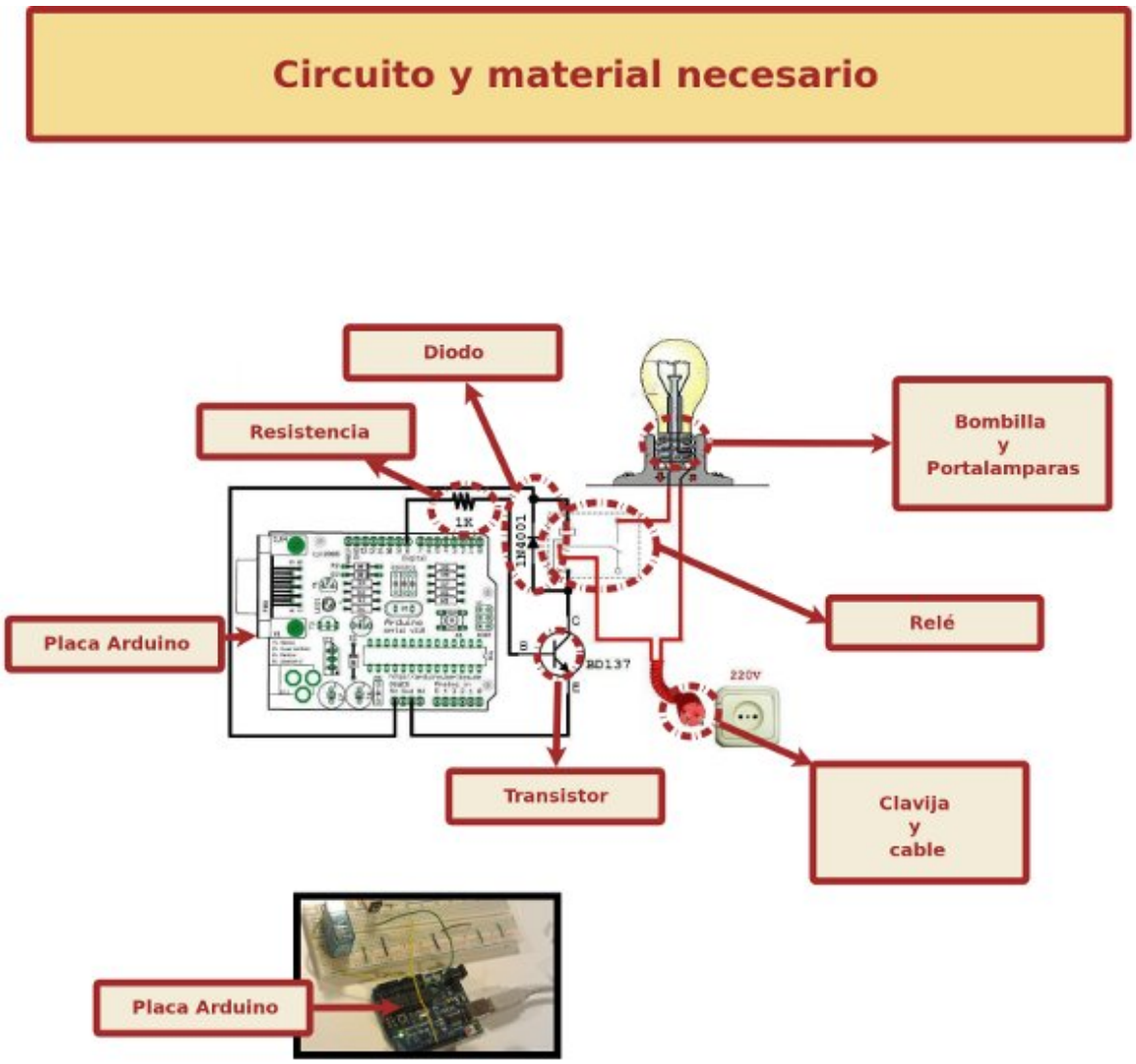
En este ejemplo mostraremos como encender una bombilla a 220V desde un lugar remoto .

Este tutorial está basado en un tutorial de la página de Arduino , ver referencias al final del post.

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro smartphone Android.

Material necesario:

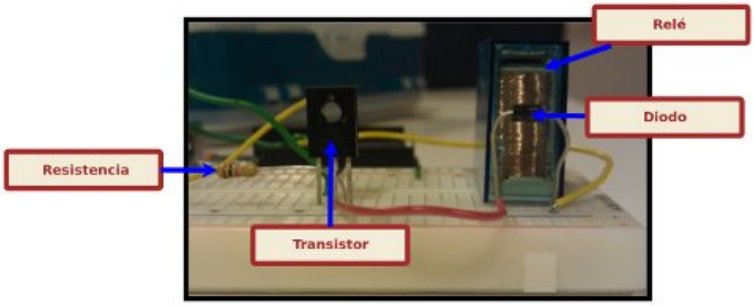
-Placa Arduino
-1 Relé de 5V ó 6V a 220V
-1 Diodo
-1 Transistor 5V
-1 Resistencia
-Cable
-1 Clavija
-1 Portalamparas
-1 Bombilla

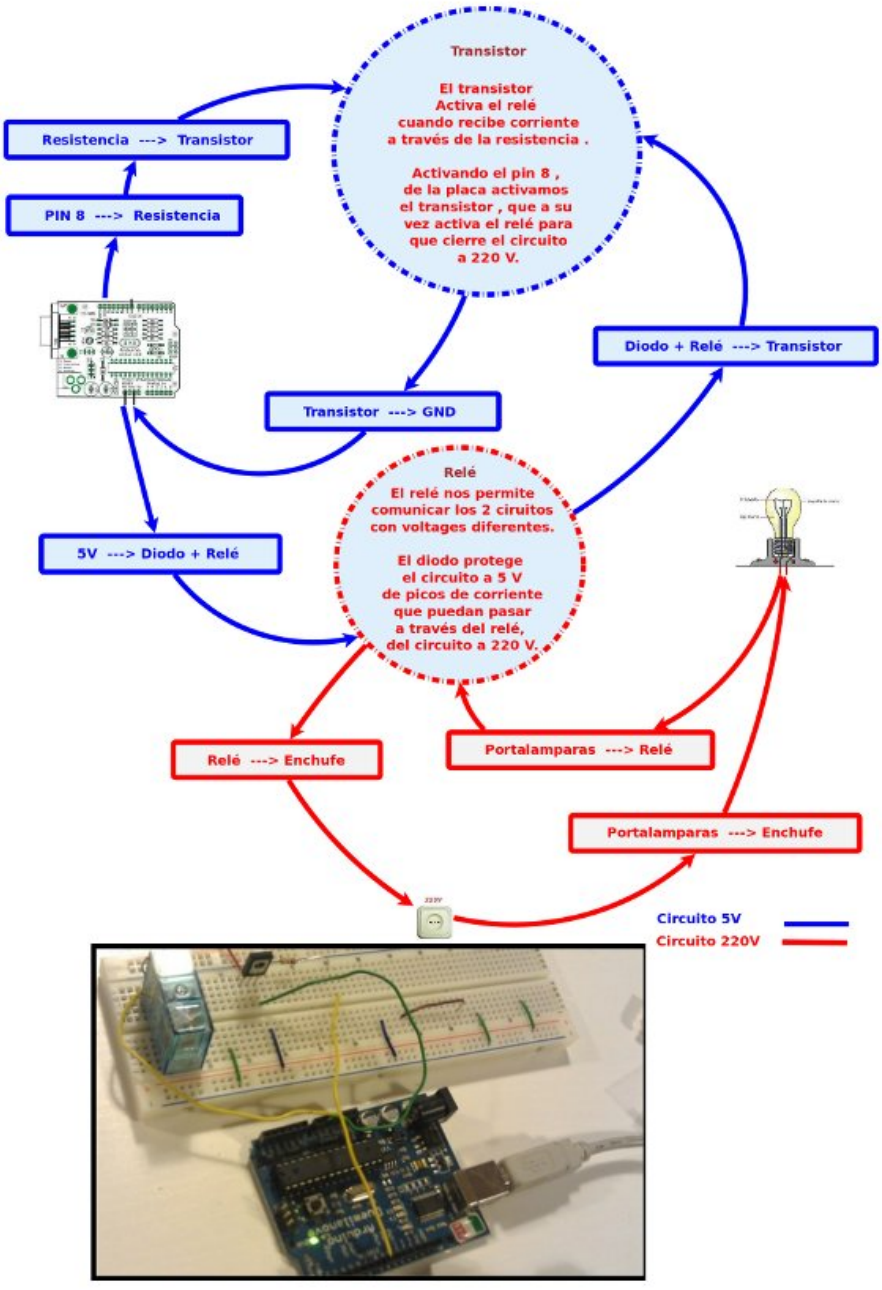


Material necesario

Una vez tengamos el material necesario procedemos a realizar el cableado de los circuitos .

Descripción del cableado de los circuitos





descripción circuitos

Una vez preparado el cableado , el código necesario para la placa Arduino es muy sencillo.
Utilizamos el pin 8 para activar el relé por medio del transistor.

Abrimos el puerto USB 115200 en modo lectura para recibir las notificaciones que nos enviará el servidor PHP.

Si por el puerto nos entra el caracter 2 , activa el relé , si es 1 lo desactiva.

Código Arduino:

```
int ledPin = 8;

int number_in = 0;

void setup() {

pinMode(ledPin, OUTPUT);

Serial.begin(115200);

}

void loop() {

if (Serial.available() > 0) {

number_in = Serial.read();

}

if (number_in > 0) {

if(number_in==2)
{
digitalWrite(ledPin, HIGH);
}
else
{
```

```
if(number_in==1)
{
digitalWrite(ledPin, LOW);
}
}

}

number_in = 0;

}
```

Utilizaremos un simple servidor php para enviar la señal a la placa Arduino conectada a nuestro servidor GNU/linux , en este caso a través del cable USB.

En nuestro caso la ruta al dispositivo que vamos a utilizar es /dev/ttyUSB0.

El puerto de la conexión USB que utilizaremos para comunicarnos con la placa Arduino será el 115200 .

Para encender o apagar la bombilla enviaremos por get a la ruta de nuestro servidor la variable hight , para encender , y la variable low para apagar.

Finalmente el servidor responde con un mensaje si ha recibido una señal correcta.

Código PHP:

```
<?php // Nonzero number to be sent to Arduino $c = 0; if(isset($_GET["hight"])) { $c=2; } if(isset($_GET["low"])) { $c=1; } if($c>0)
{
// Include the PHP serial class
require_once("phpSerial.php");

// Start a new serial class

$serial = new phpSerial;
```

```
// Specify the device being used

$serial->deviceSet("/dev/ttyUSB0");

// Set baud rate

$serial->confBaudRate(115200);

$serial->confParity("none");

$serial->confCharacterLength(8);

$serial->confStopBits(1);

$serial->confFlowControl("none");

// Open the device

$serial->deviceOpen();

// Write to the device

$serial->sendMessage(chr($c));

// Close the port

$serial->deviceClose();
$message= "Signal ".$c." Received! .
";

die(json_encode(array('status' => 'success', 'data' =>
$message)));
}
else
{
$message= "No Signal Received! .";
die(json_encode(array('status' => 'failed', 'data' =>
$message)));
}
?>
```

Y ya lo tenemos , tenemos una página php que nos funciona como

un servicio para poder encender o apagar una bombilla .

Podríamos crear en este punto una pequeña interfaz html , aprovechando el mismo fichero .php , comentando las líneas die(… , creando unos botones para encender y apagar la bombilla , pero nuestra intención es utilizarlo como servicio para una aplicación Android que os explicaremos en el próximo tutorial.

En este ejemplo hemos utilizado la librería phpSerial.php ,no recuerdo de donde la saqué así que os dejo el código , aunque en principio cualquier librería que os permita comunicar con el puerto serie debería valer.

este es el código:

```
<?php define ("SERIAL_DEVICE_NOTSET", 0); define ("SERIAL_DEVICE_SET", 1); define ("SERIAL_DEVICE_OPENED", 2); /** * Serial port control class * * THIS PROGRAM COMES WITH ABSOLUTELY NO WARANTIES ! * USE IT AT YOUR OWN RISKS ! * * @author Rémy Sanchez  * @thanks Aurélien Derouineau for finding how to open serial ports with windows
* @thanks Alec Avedisyan for help and testing with reading
* @copyright under GPL 2 licence
*/
class phpSerial
{
var $_device = null;
var $_windevice = null;
var $_dHandle = null;
var $_dState = SERIAL_DEVICE_NOTSET;
var $_buffer = "";
var $_os = "";

/**
* This var says if buffer should be flushed by sendMessage (true) or manualy (false)
*
```

```
* @var bool
*/
var $autoflush = true;

/**
* Constructor. Perform some checks about the OS and setserial
*
* @return phpSerial
*/
function phpSerial ()
{
setlocale(LC_ALL, "en_US");

$sysname = php_uname();

if (substr($sysname, 0, 5) === "Linux")
{
$this->_os = "linux";

if($this->_exec("stty --version") === 0)
{
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("No  stty  availible,  unable  to  run.",
E_USER_ERROR);
}
}
elseif(substr($sysname, 0, 7) === "Windows")
{
$this->_os = "windows";
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("Host OS is neither linux nor windows, unable tu
run.", E_USER_ERROR);
```

```
exit();
}
}

//
// OPEN/CLOSE DEVICE SECTION -- {START}
//

/**
* Device set function : used to set the device name/address.
* -> linux : use the device address, like /dev/ttyS0
* -> windows : use the COMxx device name, like COM1 (can also
be used
* with linux)
*
* @param string $device the name of the device to be used
* @return bool
*/
function deviceSet ($device)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
if ($this->_os === "linux")
{
if (preg_match("@^COM(\d+):?$@i", $device, $matches))
{
$device = "/dev/ttyS" . ($matches[1] - 1);
}

if ($this->_exec("stty -F " . $device) === 0)
{
$this->_device = $device;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}
elseif ($this->_os === "windows")
{
```

```
if (preg_match("@^COM(\d+):?$@i", $device, $matches) and
$this->_exec(exec("mode " . $device)) === 0)
{
$this->_windevice = "COM" . $matches[1];
$this->_device = "\\.\com" . $matches[1];
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}

trigger_error("Specified serial port is not valid",
E_USER_WARNING);
return false;
}
else
{
trigger_error("You must close your device before to set an
other one", E_USER_WARNING);
return false;
}
}

/**
* Opens the device for reading and/or writing.
*
* @param string $mode Opening mode : same parameter as fopen()
* @return bool
*/
function deviceOpen ($mode = "r+b")
{
if ($this->_dState === SERIAL_DEVICE_OPENED)
{
trigger_error("The device is already opened", E_USER_NOTICE);
return true;
}

if ($this->_dState === SERIAL_DEVICE_NOTSET)
{
```

```
trigger_error("The device must be set before to be open",
E_USER_WARNING);
return false;
}

if (!preg_match("@^[raw]\+?b?$@", $mode))
{
trigger_error("Invalid opening mode : ".$mode.". Use fopen()
modes.", E_USER_WARNING);
return false;
}

$this->_dHandle = @fopen($this->_device, $mode);

if ($this->_dHandle !== false)
{
stream_set_blocking($this->_dHandle, 0);
$this->_dState = SERIAL_DEVICE_OPENED;
return true;
}

$this->_dHandle = null;
trigger_error("Unable to open the device", E_USER_WARNING);
return false;
}

/**
* Closes the device
*
* @return bool
*/
function deviceClose ()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
return true;
}

if (fclose($this->_dHandle))
```

```
{
$this->_dHandle = null;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}

trigger_error("Unable to close the device", E_USER_ERROR);
return false;
}

//
// OPEN/CLOSE DEVICE SECTION -- {STOP}
//

//
// CONFIGURE SECTION -- {START}
//

/**
* Configure the Baud Rate
* Possible rates : 110, 150, 300, 600, 1200, 2400, 4800, 9600,
38400,
* 57600 and 115200.
*
* @param int $rate the rate to set the port in
* @return bool
*/
function confBaudRate ($rate)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the baud rate : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$validBauds = array (
110 => 11,
```

```
150 => 15,
300 => 30,
600 => 60,
1200 => 12,
2400 => 24,
4800 => 48,
9600 => 96,
19200 => 19,
38400 => 38400,
57600 => 57600,
115200 => 115200
);

if (isset($validBauds[$rate]))
{
if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " . (int)
$rate, $out);
}
elseif ($this->_os === "windows")
{
$ret = $this->_exec("mode " . $this->_windevice . " BAUD=" .
$validBauds[$rate], $out);
}
else return false;

if ($ret !== 0)
{
trigger_error ("Unable to set baud rate: " . $out[1],
E_USER_WARNING);
return false;
}
}
}

/**
* Configure parity.
```

```
* Modes : odd, even, none
*
* @param string $parity one of the modes
* @return bool
*/
function confParity ($parity)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set parity : the device is either not
set or opened", E_USER_WARNING);
return false;
}

$args = array(
"none" => "-parenb",
"odd" => "parenb parodd",
"even" => "parenb -parodd",
);

if (!isset($args[$parity]))
{
trigger_error("Parity mode not supported", E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
$args[$parity], $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " PARITY=" .
$parity{0}, $out);
}

if ($ret === 0)
```

```
{
return true;
}

trigger_error("Unable to set parity : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of a character.
*
* @param int $int length of a character (5 <= length <= 8) *
@return bool */ function confCharacterLength ($int) { if
($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set length of a character : the
device is either not set or opened", E_USER_WARNING);
return false;
}

$int = (int) $int;
if ($int < 5) $int = 5; elseif ($int > 8) $int = 8;

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " cs" .
$int, $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " DATA=" .
$int, $out);
}

if ($ret === 0)
{
return true;
```

```
}

trigger_error("Unable to set character length : " .$out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of stop bits.
*
* @param float $length the length of a stop bit. It must be
either 1,
* 1.5 or 2. 1.5 is not supported under linux and on some
computers.
* @return bool
*/
function confStopBits ($length)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the length of a stop bit : the
device is either not set or opened", E_USER_WARNING);
return false;
}

if ($length != 1 and $length != 2 and $length != 1.5 and
!($length == 1.5 and $this->_os === "linux"))
{
trigger_error("Specified stop bit length is invalid",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
(($length == 1) ? "-" : "") . "cstopb", $out);
}
```

```
else
{
$ret = $this->_exec("mode " . $this->_windevice . " STOP=" .
$length, $out);
}

if ($ret === 0)
{
return true;
}

trigger_error("Unable to set stop bit length : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Configures the flow control
*
* @param string $mode Set the flow control mode. Availible
modes :
* -> "none" : no flow control
* -> "rts/cts" : use RTS/CTS handshaking
* -> "xon/xoff" : use XON/XOFF protocol
* @return bool
*/
function confFlowControl ($mode)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set flow control mode : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$linuxModes = array(
"none" => "clocal -crtscts -ixon -ixoff",
"rts/cts" => "-clocal crtscts -ixon -ixoff",
```

```
"xon/xoff" => "-clocal -crtscts ixon ixoff"
);
$windowsModes = array(
"none" => "xon=off octs=off rts=on",
"rts/cts" => "xon=off octs=on rts=hs",
"xon/xoff" => "xon=on octs=off rts=on",
);

if ($mode !== "none" and $mode !== "rts/cts" and $mode !==
"xon/xoff") {
trigger_error("Invalid  flow  control  mode  specified",
E_USER_ERROR);
return false;
}

if ($this->_os === "linux")
$ret = $this->_exec("stty -F " . $this->_device . " " .
$linuxModes[$mode], $out);
else
$ret = $this->_exec("mode " . $this->_windevice . " " .
$windowsModes[$mode], $out);

if ($ret === 0) return true;
else {
trigger_error("Unable to set flow control : " . $out[1],
E_USER_ERROR);
return false;
}
}

/**
* Sets a setserial parameter (cf man setserial)
* NO MORE USEFUL !
* -> No longer supported
* -> Only use it if you need it
*
* @param string $param parameter name
* @param string $arg parameter value
```

```
* @return bool
*/
function setSetserialFlag ($param, $arg = "")
{
if (!$this->_ckOpened()) return false;

$return = exec ("setserial " . $this->_device . " " . $param .
" " . $arg . " 2>&1");

if ($return{0} === "I")
{
trigger_error("setserial: Invalid flag", E_USER_WARNING);
return false;
}
elseif ($return{0} === "/")
{
trigger_error("setserial:  Error  with  device  file",
E_USER_WARNING);
return false;
}
else
{
return true;
}
}

//
// CONFIGURE SECTION -- {STOP}
//

//
// I/O SECTION -- {START}
//

/**
* Sends a string to the device
*
* @param string $str string to be sent to the device
```

```
* @param float $waitForReply time to wait for the reply (in
seconds)
*/
function sendMessage ($str, $waitForReply = 0.1)
{
$this->_buffer .= $str;

if ($this->autoflush === true) $this->flush();

usleep((int) ($waitForReply * 1000000));
}

/**
* Reads the port until no new datas are availible, then return
the content.
*
* @pararm int $count number of characters to be read (will
stop before
* if less characters are in the buffer)
* @return string
*/
function readPort ($count = 0)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened to read it",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$content = ""; $i = 0;

if ($count !== 0)
{
do {
if ($i > $count) $content .= fread($this->_dHandle, ($count -
```

```
$i));
else $content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}
else
{
do {
$content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}

return $content;
}
elseif ($this->_os === "windows")
{
/* Do nohting : not implented yet */
}

trigger_error("Reading serial port is not implemented for
Windows", E_USER_WARNING);
return false;
}

/**
* Flushes the output buffer
*
* @return bool
*/
function flush ()
{
if (!$this->_ckOpened()) return false;

if (fwrite($this->_dHandle, $this->_buffer) !== false)
{
$this->_buffer = "";
return true;
}
else
```

```
{
$this->_buffer = "";
trigger_error("Error while sending message", E_USER_WARNING);
return false;
}
}

//
// I/O SECTION -- {STOP}
//

//
// INTERNAL TOOLKIT -- {START}
//

function _ckOpened()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened", E_USER_WARNING);
return false;
}

return true;
}

function _ckClosed()
{
if ($this->_dState !== SERIAL_DEVICE_CLOSED)
{
trigger_error("Device must be closed", E_USER_WARNING);
return false;
}

return true;
}

function _exec($cmd, &$out = null)
{
```

```
$desc = array(
1 => array("pipe", "w"),
2 => array("pipe", "w")
);

$proc = proc_open($cmd, $desc, $pipes);

$ret = stream_get_contents($pipes[1]);
$err = stream_get_contents($pipes[2]);

fclose($pipes[1]);
fclose($pipes[2]);

$retVal = proc_close($proc);

if (func_num_args() == 2) $out = array($ret, $err);
return $retVal;
}

//
// INTERNAL TOOLKIT -- {STOP}
//
}
?>
```

Referencias:

[Tutorial de la página de Arduino]

[Vídeo del turorial]

Próximamente:

Tutorial Arduino parte 4

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro celular Android.

Arduino Segunda Parte

Introducción

En este segundo post vamos a realizar una pequeña aplicación que controla dos leds (uno rojo y uno verde) conectados a Arduino. Esta placa estará conectada a un pc con un servidor servidor web. En dicho servidor web alojaremos un archivo PHP que se encargará de enviar valores a través de USB al arduino. Con un pequeño programa en Android atacaremos mediante Post a ese fichero. De esta manera podremos controlar los leds desde nuestro terminal, siempre y cuando estemos en la misma red local que el servidor web.

- GNU\Linux, venditos ficheros

GNU\Linux es un sistema operativo que utiliza el núcleo Linux junto con bibliotecas y herramientas del proyecto GNU y de muchos otros proyectos/grupos de software.
Bien, en GNU\Linux el hardware esta asignado a un fichero. Así pues, solo tendremos que esribir en el fichero que identifica al puerto USB donde tenemos conectado nuestro Arduino, ni drivers ni complicaciones. Abrir en modo escritura, escribir i

cerrar el fichero, así de fácil.

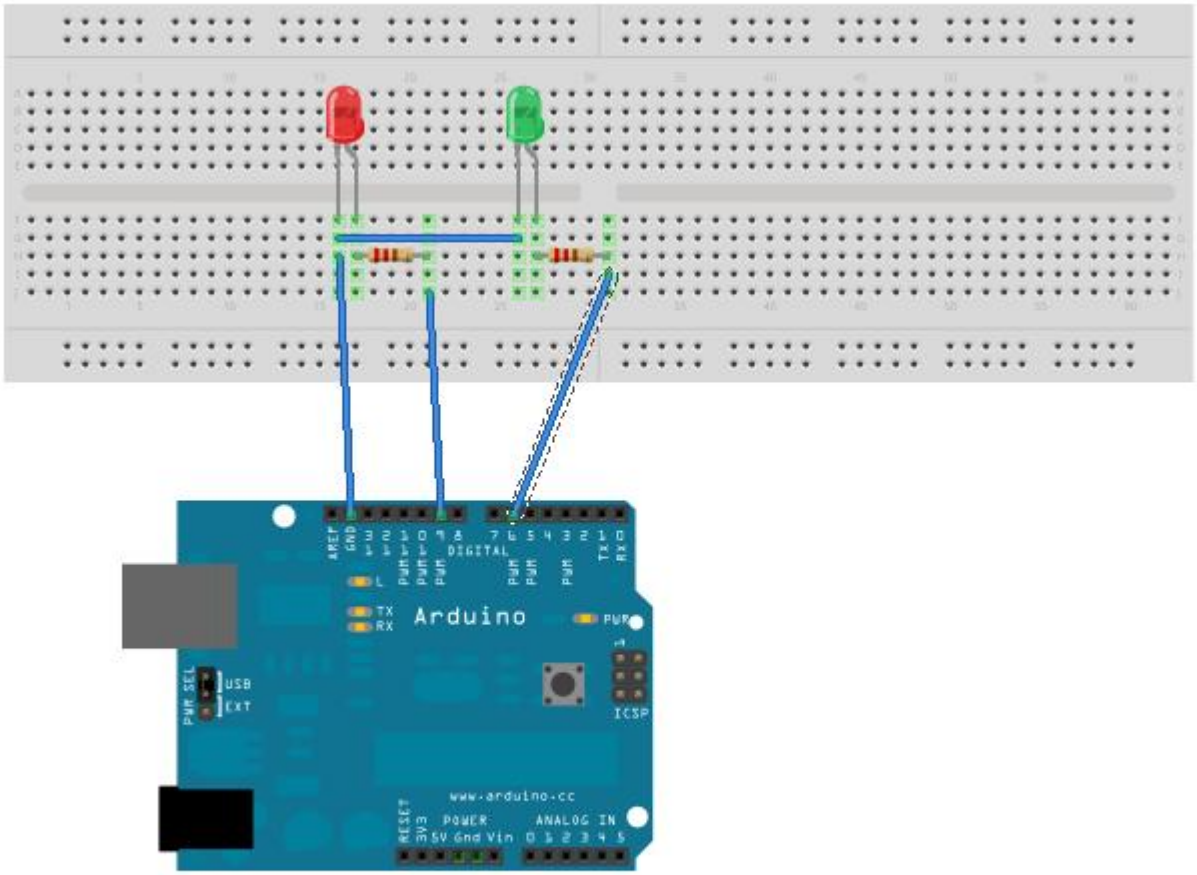
- Android

Android es un sistema operativo basado en el núcleo Linux diseñado originalmente para dispositivos móviles, tales como teléfonos inteligentes, pero que posteriormente se expandió su desarrollo para soportar otros dispositivos tales como tablet, reproductores MP3, netbook, PC, televisores, lectores de e-book e incluso, se han llegado a ver en el CES, microondas y lavadoras. Crearemos una pequeña aplicación que enviará un post a un fichero php que se encargará de escribir en el puerto USB correspondiente. Por la simplicidad de la aplicación y no ser Android objetivo del post sólo pondremos el código relativo a la ejecución del post.

Instalación de paquetes necesarios

Deberemos instalar las librerías y el IDE. Para ello podeis consultar la primera parte de la serie Arduino Arduino Primera Parte

Diseño electrico del Arduino



Código Arduino

```
int ledPin = 9;
int ledPin2 = 6;
int number_in = 0;

void setup() {
pinMode(ledPin, OUTPUT);
pinMode(ledPin2, OUTPUT);

Serial.begin(115200);
}

void loop() {
if (Serial.available() > 0) {
```

```
number_in = Serial.read();
Serial.print("He rebut: ");
Serial.println(number_in, DEC);
}

while (number_in == 49) {
digitalWrite(ledPin, HIGH);
delay(800);
digitalWrite(ledPin, LOW);
delay(800);
if (Serial.available() > 0) {
number_in = Serial.read();
Serial.print("He rebut: ");
Serial.println(number_in, DEC);
}
}

while (number_in == 51) {
digitalWrite(ledPin2, HIGH);
delay(800);
digitalWrite(ledPin2, LOW);
delay(800);
if (Serial.available() > 0) {
number_in = Serial.read();
Serial.print("He rebut: ");
Serial.println(number_in, DEC);
}
}
}
```

Servidor Web

- Deberemos dar permisos al usuario www-data (usuario por defecto del servidor Apache) para escribir en el fichero que corresponde al puerto USB:

```
sudo adduser www-data dialout
```

- En el servidor web alojaremos el fichero que se encargará de enviar las señales a Arduino:

```
if (!empty($_POST['green'])) {
$return['msg'] = "green";
// '/dev/ttyACM1' corresponde a al puerto USB deberemos cambiarlo por el que corresponda en nuestro PC
$fp =fopen("/dev/ttyACM1", "w");
fwrite($fp, chr('51'));
fclose($fp);
}else if (!empty($_POST['red'])){
$return['msg'] = "red";
$fp =fopen("/dev/ttyACM1", "w");
fwrite($fp, chr('49'));
fclose($fp);
}else if (!empty($_POST['off'])){
$return['msg'] = "off";
$fp =fopen("/dev/ttyACM1", "w");
fwrite($fp, chr('48'));
fclose($fp);
}
```

Código Android

- Encender led Verde

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://localhost/post.php");

List postValues = new ArrayList(2);
postValues.add(new BasicNameValuePair("green", "green"));
httppost.setEntity(new UrlEncodedFormEntity(postValues));
```

```
HttpResponse response = httpclient.execute(httppost);
```

- Encender led Rojo

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://localhost/post.php");

List postValues = new ArrayList(2);
postValues.add(new BasicNameValuePair("red", "red"));
httppost.setEntity(new UrlEncodedFormEntity(postValues));

HttpResponse response = httpclient.execute(httppost);
```

- Apagar leds

```
HttpClient httpclient = new DefaultHttpClient();
HttpPost httppost = new HttpPost("http://localhost/post.php");

ListpostValues = new ArrayList(2);
postValues.add(new BasicNameValuePair("off", "off"));
httppost.setEntity(new UrlEncodedFormEntity(postValues));

HttpResponse response = httpclient.execute(httppost);
```

Observaciones

Con esta segunda parte podemos ver una aproximación a las posibilidades de arduino. De esta manera podemos controlar cualquier cosa que connectemos a Arduino (las posibilidades son infinitas) y es que con este «juguete» se sabe cuando se empieza pero no cuando se acaba.

Fuentes

- Wikipedia – Android
- Android Hello World

Ruben

Arduino – Primera Parte

Introducción
En este primer post mostraremos que es Arduino, como funciona y como cargar el programa Blink. Esto nos permitirá ver el funcionamiento básico y comprobar que nuestra placa funciona correctamente.

- Arduino 

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.
El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (boot loader) que corre en la placa.
Arduino se puede utilizar para desarrollar objetos

interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.
Al ser open-hardware, tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

- Processing

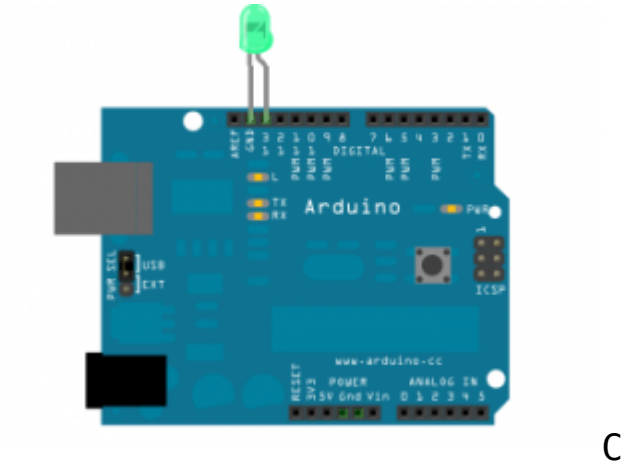
Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue iniciado por Ben Fry y Casey Reas a partir de reflexiones en el Aesthetics and Computation Group del MIT Media Lab dirigido por John Maeda.
Processing es desarrollado por artistas y diseñadores como una herramienta alternativa al software propietario. Puede ser utilizado tanto para aplicaciones locales así como para

aplicaciones web (Applets).
Se distribuye bajo la licencia GNU GPL.

Instalando el software necesario

Deberemos instalar las librerías y el IDE con este simple comando:
sudo apt-get install arduino arduino-core

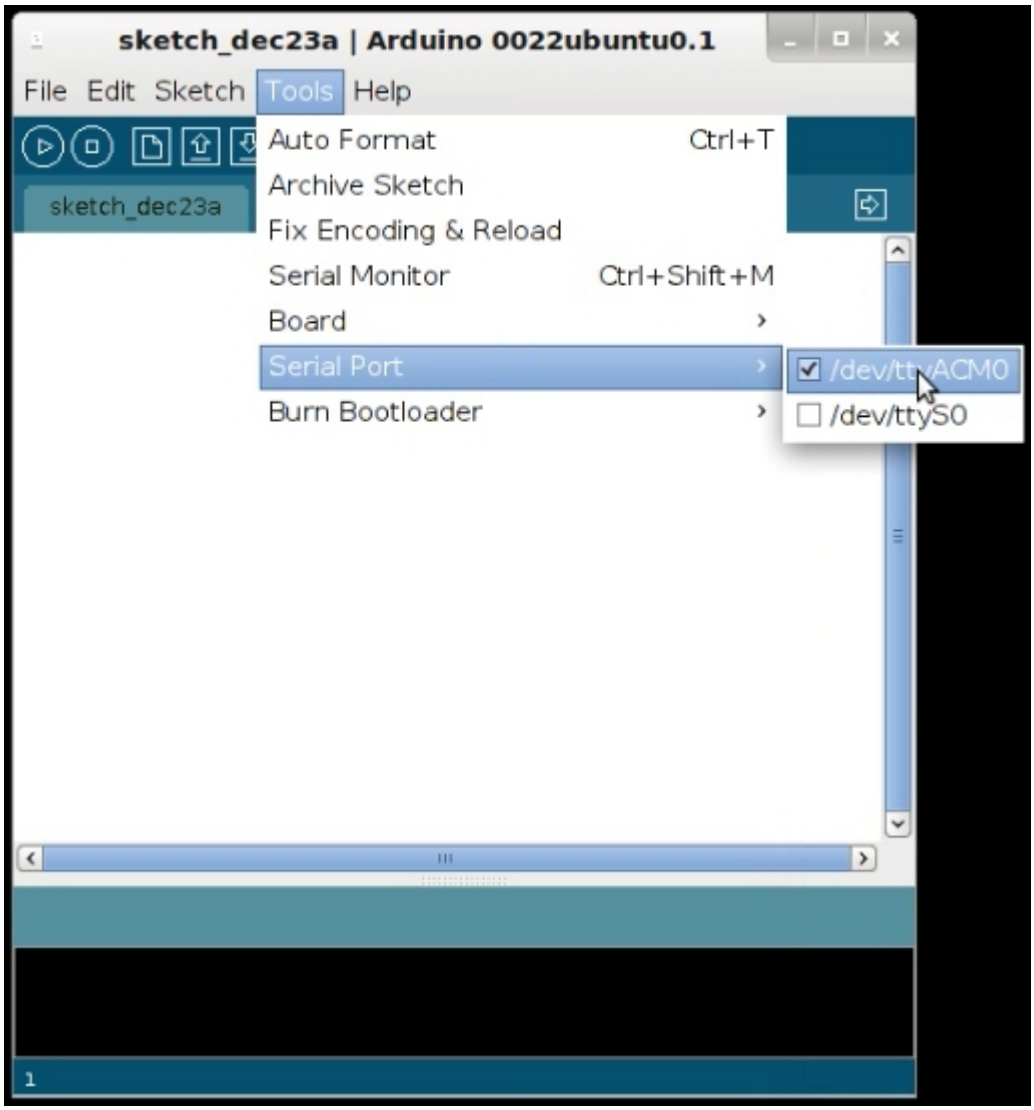
Diseño Arduino



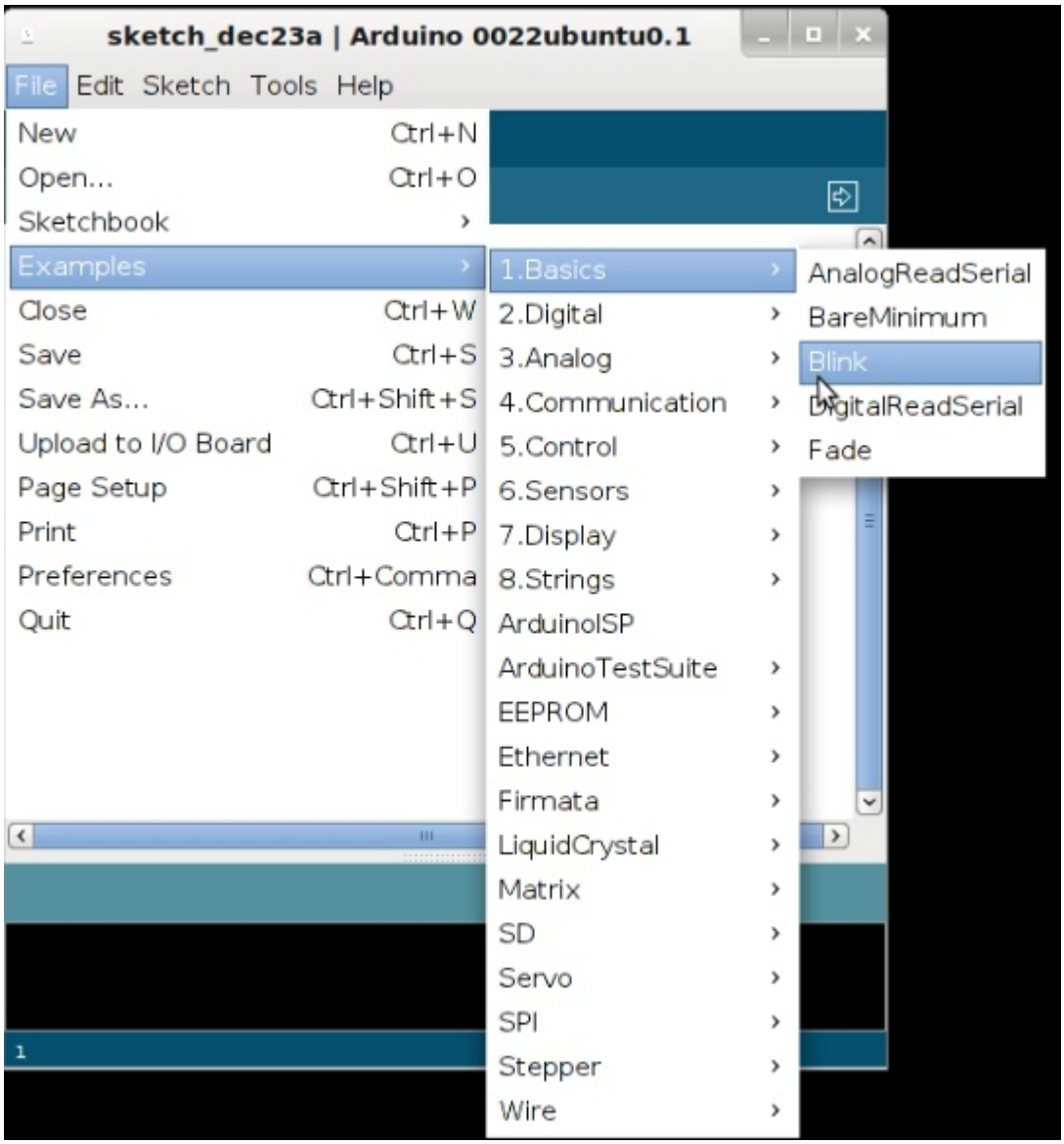
Conectaremos el led teniendo en cuenta que la patilla corta es el negativo y la patilla larga el positivo. Necesitaríamos una resistencia para que no se queme el led, por suerte el pin 13 de la placa Arduino incluye una. Así pues, conectaremos la patilla larga(positivo) al conector 13 y la corta(negativo) a la entrada de corriente GND, como se puede comprobar los dos conectores y la resistencia están puestos estrategicamente.

Cargando el programa Blink

Este programa es el más básico, simplemente produce un parpadeo del led. Conectamos el Arduino por USB. Deberemos comprobar que hemos seleccionado correctamente el puerto USB, en mi caso es el /dev/ttyACM0 tal y como se ve en la imagen.



Cargaremos el programa Blink.



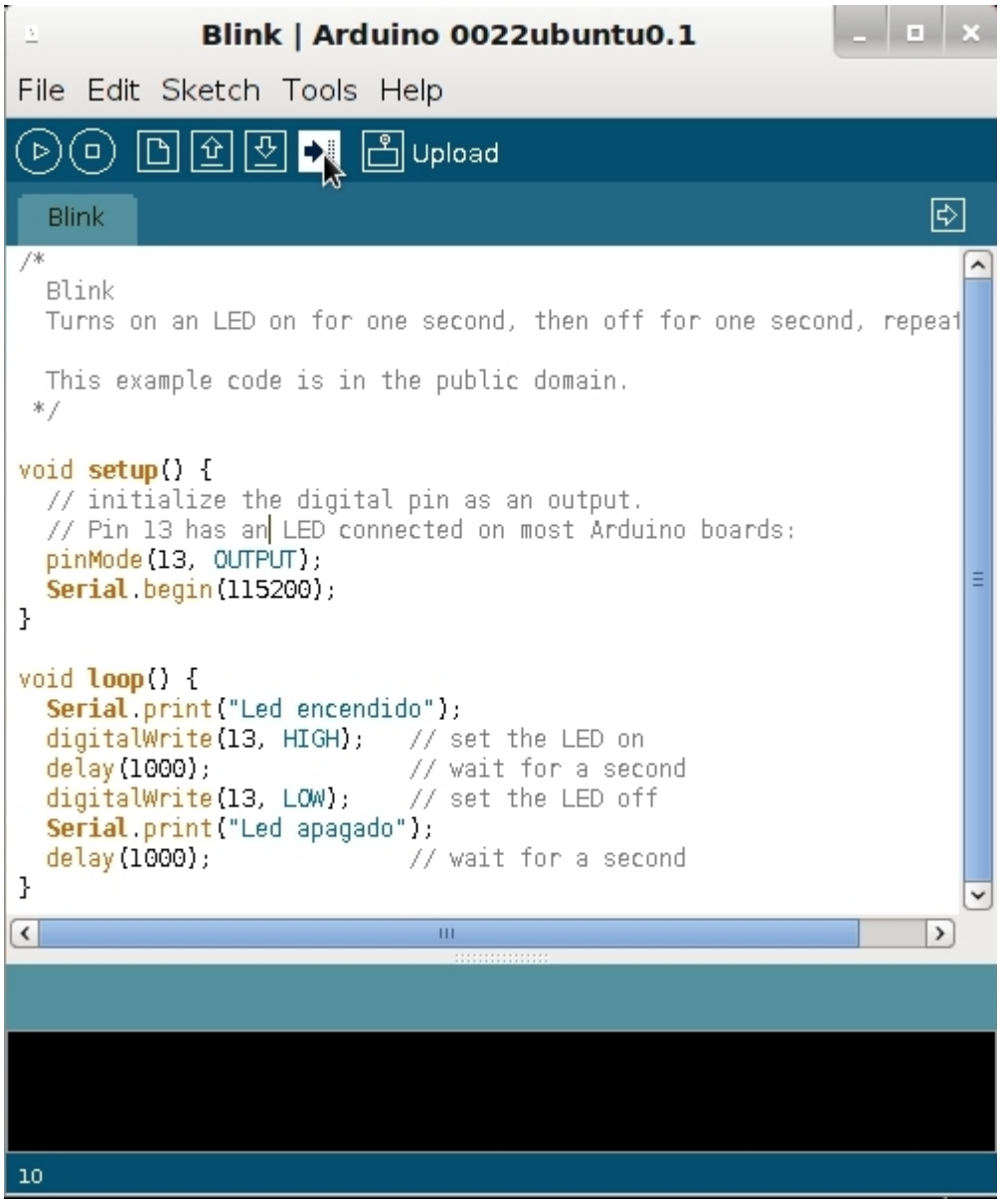
Haremos unas pequeñas modificaciones para poder obtener feedback de lo que esta ocurriendo. Processing nos dirá que el código es read-only, simplemente guardamos el código con un nuevo nombre y listo.

```
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
```

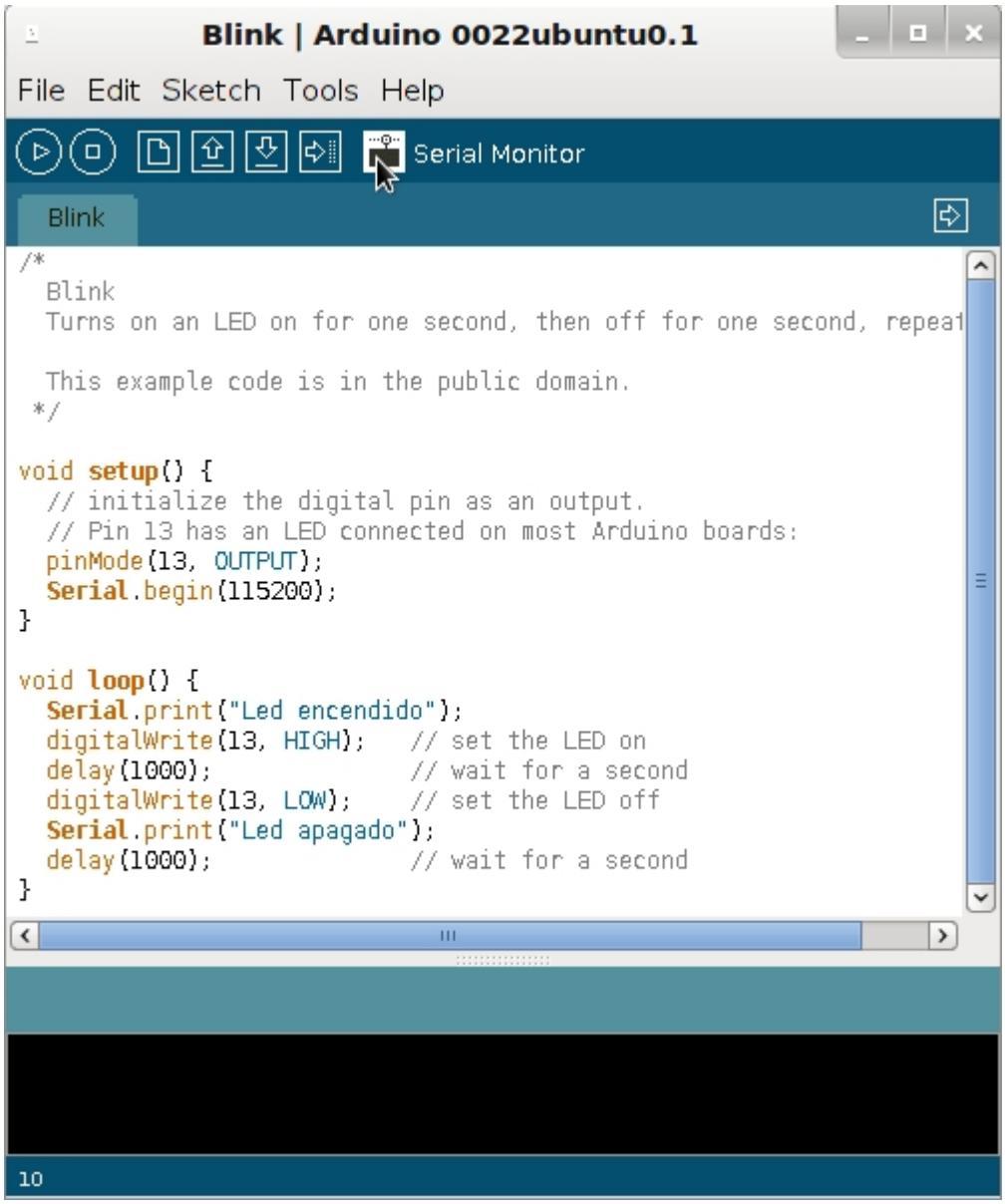
```
This example code is in the public domain.
*/

void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
Serial.begin(115200); //Establecemos la velocidad de envio de
datos
}

void loop() {
Serial.print("Led encendido");
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
Serial.print("Led apagado");
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
}
```

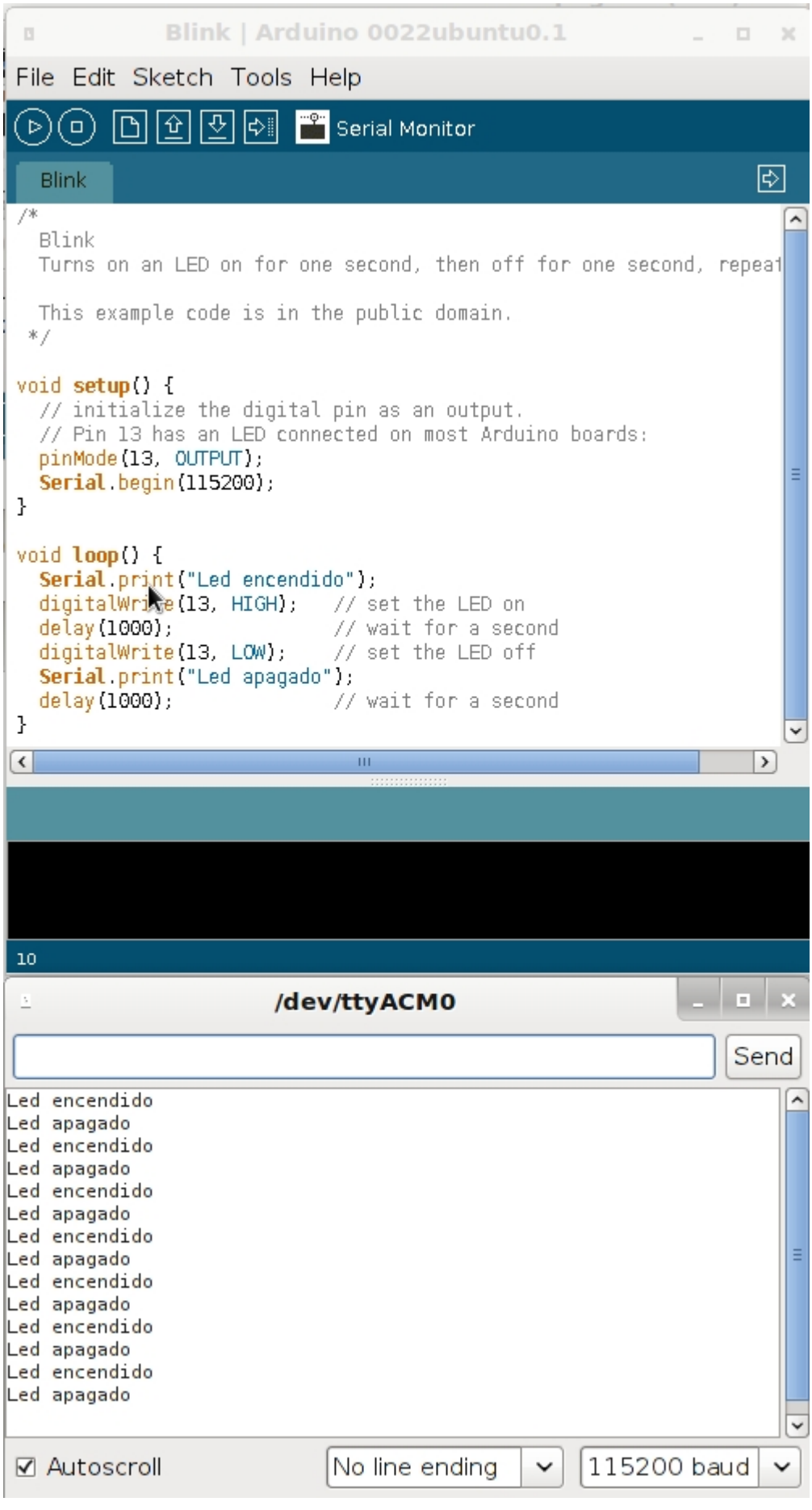
Subir el programa a Arduino y arrancar el Serial Monitor

Subiremos el programa tal y como se ve en la imagen.



Una vez cargado el programa arrancamos el Serial Monitor y podremos ver el feedback que recibimos del Arduino al mismo tiempo que nuestro led empieza a parpadear. Debemos tener en cuenta que la velocidad que hemos indicado en el codigo Serial.begin(115200) sea la misma con la que escucha el Serial monitor(Desplegables inferior derecho).


Blink | Arduino 0022ubuntu0.1
File Edit Sketch Tools Help
Serial Monitor
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeat
This example code is in the public domain.
*/
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
Serial.begin(115200);
}
void loop() {
Serial.print("Led encendido");
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
Serial.print("Led apagado");
delay(1000); // wait for a second
}
10


Blink | Arduino 0022ubuntu0.1
File Edit Sketch Tools Help
Serial Monitor
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeat
This example code is in the public domain.
*/
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
Serial.begin(115200);
}
void loop() {
Serial.print("Led encendido");
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
Serial.print("Led apagado");
delay(1000); // wait for a second
}
10
/dev/ttyACM0
Send
Led encendido
Led apagado
Led encendido
Led apagado
Led encendido
Led apagado
Led encendido
Led apagado
Led encendido
Led apagado
Led encendido
Led apagado
Led encendido
Led apagado
Autoscroll
No line ending
115200 baud

Observaciones

Con este primer post podemos tener un primer contacto con este espectacular juguete. Las posibilidades son muchas y se pueden encontrar miles de proyectos en la red. Próximamente, mostraré un proyecto un poco más avanzado son Arduino y más adelante el compañero Scuraki continuará con la tercera parte.

Fuentes

- Arduino Blink
- Wikipedia – Arduino

Ruben

Android sdk en Ubuntu unknown device ???????????

Tras instalar android sdk , eclipse , actualizar etc , nos podemos encontrar con que el emulador funciona,
pero no podemos debugar en el dispositivo.

Debugar en el dispositivo , nos ayuda a crear aplicaciones para los sensores , y nos permite probarlo al mismo tiempo , como los sensores de nivel , gps , etc , sin el dispositivo sería muy difícil .

Problema en DDMS , dispositivo desconocido ???????????
No funciona el debug con el dispositivo.

/android-sdk/platform-tools$ ./adb devices devuelve :

List of devices attached

???????????? ??
unknown device ????????????
[Fuente de la solución]
En esta zona podemos ver la solución:

> *14:25 W/ddms: Unable to get frame buffer: device (????????????) request*
> *rejected: insufficient permissions for device*
> ==
>
> *I then checked*
>
> ==
> *$ ./adb devices*
> *List of devices attached*
> *???????????? no permissions*
> ==
>
> *There appears to be a permission problem. I then wrote a*
> *`51-android.rules' file (with chmod a+r) in /etc/udev/rules.d/, such that*
>
> ==
> *# cat 51-android.rules*
> *SUBSYSTEM==»usb», ATTRS{idVendor}==»0bb4″, ATTRS{idProduct}==»0c87″,*
> *MODE=»0666″*
> ==
>
> *(the idVendor is the HTC one). I then restarted udev, but to no avail:*
> *same problems. Any idea?*
>
>
>

_

Merciadri Luca
See http://www.student.montefiore.ulg.ac.be/~merciadri/
I use PGP. If there is an incompatibility problem with your mail
client, please contact me.

Old 08-13-2010, 02:33 PM
Merciadri Luca

Default Android phone is not recognized by Android SDK, despite udev conf
Problem solved: restarting udev from CLI was not sufficient. I needed to
restart.

Resumiendo:
1:)-> crear el fichero /etc/udev/rules.d/51-android.rules :
$ sudo gedit /etc/udev/rules.d/51-android.rules
con el siguiente contenido:
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bb4", ATTRS{idProduct}=="0c87",
MODE="0666"

2:)-> darle permisos $ sudo chmod a+r /etc/udev/rules.d/51-android.rules
3:)-> reiniciar

Bueno , ya podemos utilizar el dispositivo , en la carpeta donde esté instalado adb ,
./adb devices , devuelve en mi caso

List of devices attached
HT0C4RX21596 device

ya podemos debugar , probamos creando un proyecto Android en eclipse , pulsamos el botón de debug y ya tenemos nuestro hello world …. que tanto deseábamos.