engduino® Tutorial

ENGDUINO SUPPORT TEAM - SUPPORT@ENGDUINO.ORG

Introduction

The Engduino is just a simple, small, computer that is capable of measuring quite a lot of interesting things about the real world and that has a bunch of LEDs (lights) that can be used to provide feedback to the user.

This is a first hands-on tutorial, intended for people who have never used similar computers before or, for that matter, done any programming. It is a step-by-step guide to creating a very simple program and watching that run. As we hope you'll find, this is quite an exciting thing to do. And the good news is that it doesn't stop there – as you improve in programming ability, you can also build really quite sophisticated things with the Engduino.

Overview of the Engduino Board

For those who know about these things, the Engduino is, at heart, an Arduino (just a brand of simple computer), and it is programmed using a slightly modified version of standard Arduino software. We are going to write a very simple app that we want our Engduino to execute and then, using a USB cable and special communication software; we will upload it to our Engduino and see what happens.

Your Engduino board comes pre-fitted with some sensors and LEDs so we can get straight into experimenting with programming the board without needing to do any construction beforehand.

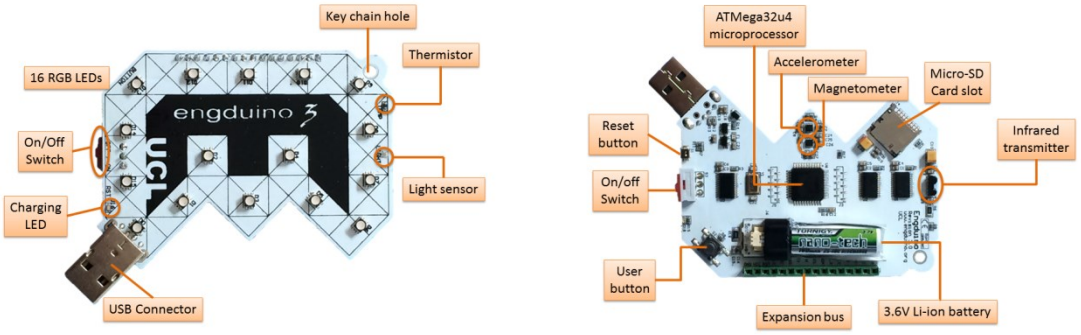


Figure 1: The top and bottom of your Engduino

On the one side of your Engduino you have 16 LED lights that have adjustable brightness and can be set to various mixtures of the three primary colours (of light): Red, Green and Blue.

Figure 2: 16 Red, Green, Blue LEDs on one side of the board

The LEDS are numbered 0 – 15 and their exact positions can be seen in Appendix A. Also on this side of the board we have a thermistor which can be used to detect temperature and a light sensor. On the other side we have a protected Lithium polymer (Lipo) battery, the same type of batteries that are in mobile phones and tablets.

Important Battery Safety Information:

(1) Follow all instructions. (2) Do not submerge Engduino® in water. (3) Do not expose Engduino® to excessive heat or place near combustion sources such as stoves.

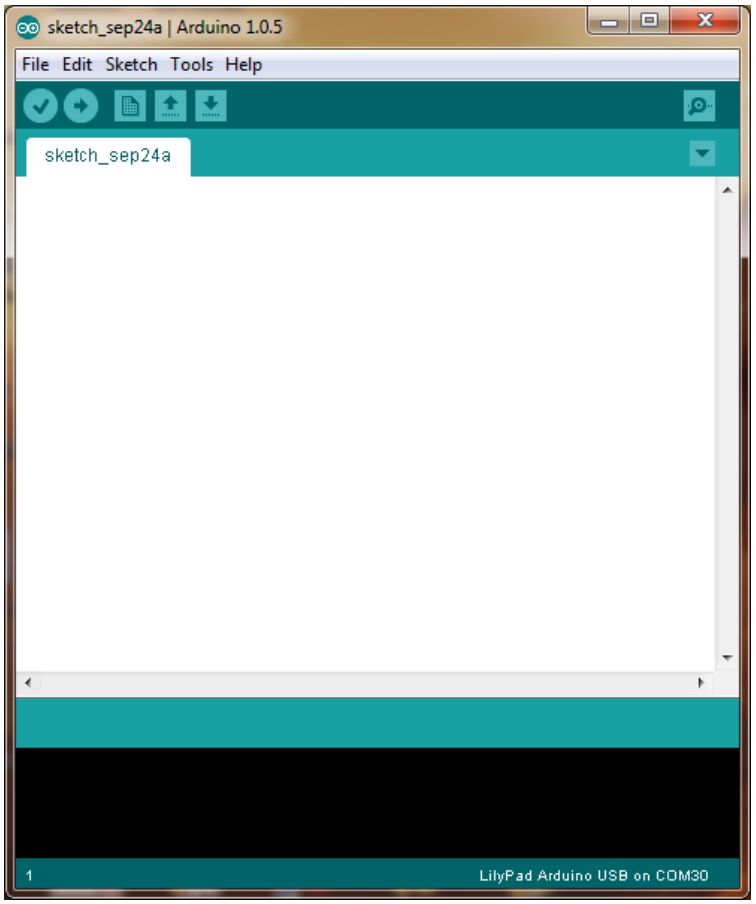
The battery charges automatically if it is plugged in while the Engduino is plugged into a USB port. There is also a button that can be used to get input from the user and a power switch to power your device down when you are not using it. For more advanced users, the accelerometer is near the keyring hole along with a magnetometer and infra-red transceiver. We will not use them in this simple tutorial, but you can find more information about them at:

http://www.engduino.org/tutorials/worksheets_pdf/

Your first app

For your first exercise you are going to write a program in your Arduino IDE and then upload it to your Engduino board.

If this is a lab class, your Arduino IDE should be open for you already. If not, just double click on the blue Arduino icon on your computer. You should see a window like that shown below:



Now we're going to write a program.

Type the following into the white box.

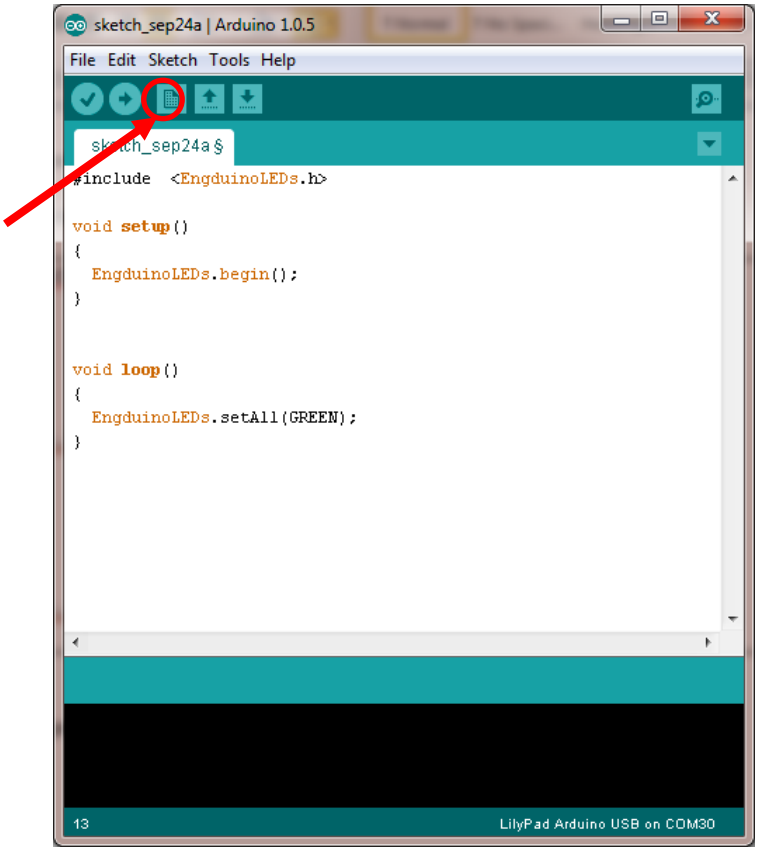
```
#include  <EngduinoLEDs.h>

void setup()
{
  EngduinoLEDs.begin();
}

void loop()
{
  EngduinoLEDs.setAll(GREEN);
}
```

Notice that `All` is spelt with an upper case `A` and lower case `ll`.

Now let's make it run…. make sure your Engduino is plugged in[1].

[1] If you are not in a lab class, make sure that in **Tools > Board > LilyPad Arduino USB** is selected, and that under **Tools > Serial Port** the serial port that's connected to your board is selected.

Click on the upload button:



This checks your code for some types of error, turns it into something that a machine can understand, and then uploads it to the Engduino board. If there are errors they will appear in the black section in red.

Voila – your first Engduino program.

Looking a bit closer

Now let's take a quick look at some interesting parts of the basic app code and see how we can customise, personalise or enhance it.

```
#include <EngduinoLEDs.h>
```

This lets your program use the LEDs. If you wanted to use the button you would have change this to be `#include <EngduinoButton.h>`, and if you wanted to use both, you would need to have both there. There is at least one #include for each of the sensors on the Engduino.

The next part of the program is:

```
void setup()
{
  EngduinoLEDs.begin();
}
```

This is a function. A function is similar to the ones you may have experienced in mathematics. It can take some value/s and returns another value after it executes. **void** is what the function returns (nothing in this case), then comes the function name (**setup** in this case) and then brackets that contain the values we pass to the function (none in this case, so the brackets are empty.) The curly brackets delimit our code, so in a file with many functions everything within the set of curly brackets just after our function name belongs to that function. Our function **setup()** has just one line

```
  EngduinoLEDs.begin();
```

This line tells the **EngduinoLEDs** object to run the function **begin()**, which gets the LEDs started so we can play with them. The semi-colon at the end of the line is common to many programming languages and signals to the computer that is the end of one instruction. Don't forget these when you start writing your own code[2]. The setup function runs exactly once, when the program starts.

The next function:

```
void loop()
{
  EngduinoLEDs.setAll(GREEN);
}
```

is called **loop()**. The loop function is run repeatedly – and it's where we put most of the instructions we want our Engduino to perform. In this case, it's just one thing:

```
  EngduinoLEDs.setAll(GREEN);
```

This line tells the **EngduinoLEDs** object to execute the function **setAll()**. We pass this function a value, namely what colour to make the LEDs (green in this case). It doesn't return a value but simply does what you tell it, and switches the LEDs on.

This line will actually be executed over and over as fast as the Engduino can go – but since it always does the same thing, you won't see any change.

Personalise Your Sketch.

1. Change the colour

The function **setAll()** can be given a number of different values including: RED, GREEN, and BLUE. Go ahead and delete the word GREEN inside the brackets and replace it with the colour of your choice all in capital letters. For example if you prefer red, your app will look like this.

```
#include  <EngduinoLEDs.h>

void setup()
{
  EngduinoLEDs.begin();
}
```

[2] You will, everyone does on occasion.

```
void loop()
{
  EngduinoLEDs.setAll(RED);
}
```

When you've done that, click the upload button, as before, to upload your code. The LEDs will now be lit according to the colour you choose. Well done!

2. Blink with Delay

The Engduino has a function built in called **delay()**. This function takes a number which tells the board to wait those number of milliseconds (remember a millisecond is one thousandth of a second) before executing the next instruction in your program. We are going to use this useful little function to make our Engduino blink.

Just below the line

```
  EngduinoLEDs.setAll(RED);
```

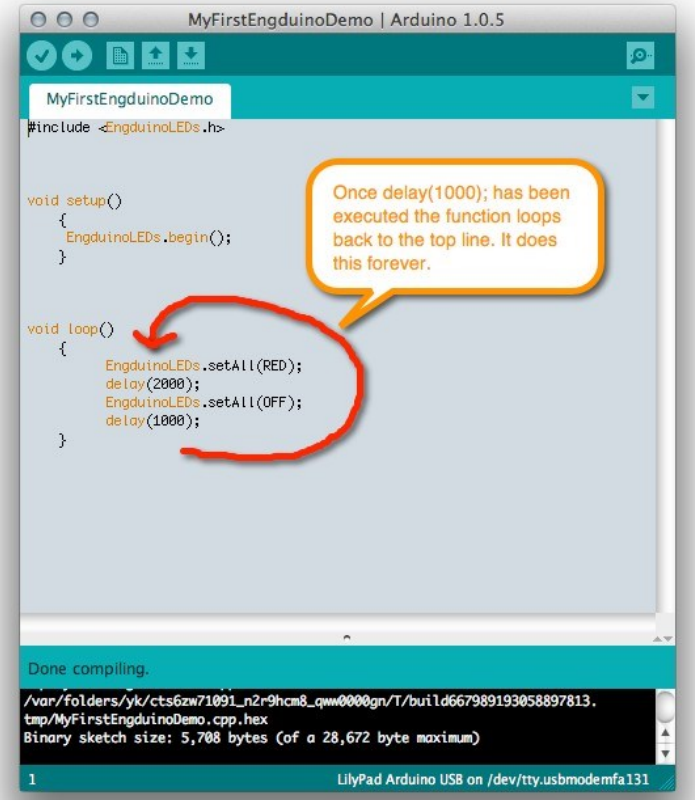
add the lines

```
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
```

Don't forget your semi-colons! Your code should now look something like this. (With the colour you chose instead of RED if you chose something else.)

```
#include  <EngduinoLEDs.h>

void setup()
{
  EngduinoLEDs.begin();
}

void loop()
{
  EngduinoLEDs.setAll(RED);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
}
```

Go ahead and program the Engduino with your code. You should see your Engduino now rhythmically blinking the LEDs in the colour that you chose (remember that the loop code is executed repeatedly).



Go ahead and experiment with the timings. Change the numbers inside the brackets of the delay functions in both the lines

```
  delay(2000);

  delay(1000);
```

to some number between 200 and 5000 or 0.2 seconds to 5 seconds - and see what happens. By playing around with combinations of numbers you can modify the sequence the blinking takes.

If we were to add the lines

```
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
```

to the bottom so our code would look like this

```
void loop()
{
  EngduinoLEDs.setAll(RED);
```

```
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
}
```

What do you think would happen? Try it and find out… what happens if you change the timings now? What happens if you make them very small?

Traffic Lights

Bravo if you've got here. You now know enough to make your own traffic light!

If you think about it a traffic light does almost the same thing as our app except it changes three colours and the timings are slightly different. We can make a couple more modifications and ours will do the same. Let's add the lines:

```
  EngduinoLEDs.setAll(GREEN);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
```

after the last line but still inside the curly brackets. Then change the sequence so that the Lights go from RED to YELLOW to GREEN and then finally back to YELLOW. Your code should look like this:

```
void loop()
{
  EngduinoLEDs.setAll(RED);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(GREEN);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
}
```

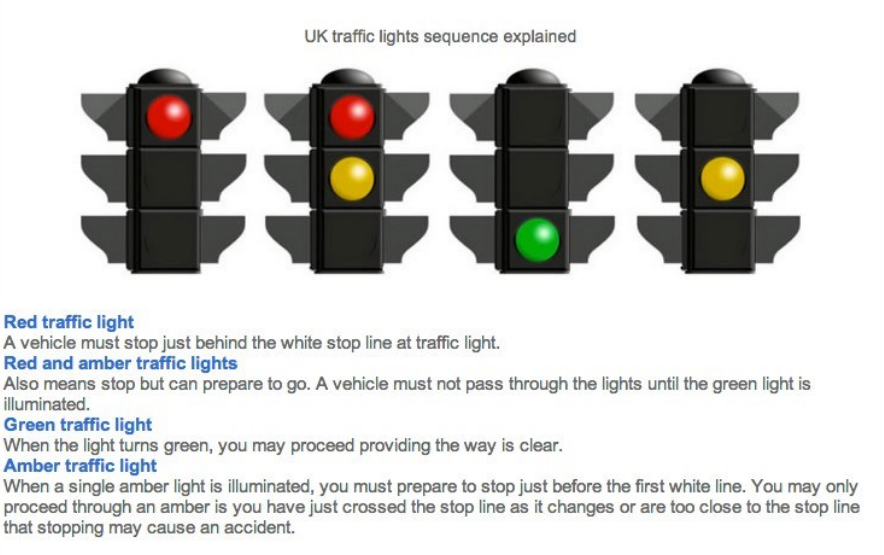
Once you're sure your code looks like this. You can go ahead and upload it with the right arrow on

the Arduino IDE.

You'll see your Engduino will begin to look a traffic light but the timing seems all wrong. Experiment with different timings to see if you can get it to look like a traffic light.

Making it more accurate

If you still have time, let's change the lighting so it more accurately reflects UK light signals. If you remember, UK traffic signals have the following pattern



So when our traffic light changes from red to the next phase we want our Engduino to show half red lights and half yellow. This is quite easy. We just have to write a little more code because we will have to set individual LEDs to colours rather than all of them with a single line. The function **EngduinoLEDs.setLED()** takes two values (separated by a comma). The first is the number of the LED you want to access and the other is the colour you want that LED to be. So for example, if I wanted to switch LED number 0 (we start counting at zero) to yellow the code would look like this.

```
  EngduinoLEDs.setLED(1, YELLOW);
```

Remember the Engduino has 16 LEDs so perhaps if we set 0 – 7 to Red and 8 – 15 to Yellow, we might make it look how we want it to.

Replace the first of the lines that look like:

```
  EngduinoLEDs.setAll(YELLOW);
```

with the following lines. (Don't forget the copy and paste functionality in the Arduino **Edit** menu item to help save you some time)

```
  EngduinoLEDs.setLED(0, RED);
  EngduinoLEDs.setLED(1, RED);
  EngduinoLEDs.setLED(2, RED);
```

```
  EngduinoLEDs.setLED(3, RED);
  EngduinoLEDs.setLED(4, RED);
  EngduinoLEDs.setLED(5, RED);
  EngduinoLEDs.setLED(6, RED);
  EngduinoLEDs.setLED(7, RED);
  EngduinoLEDs.setLED(8, YELLOW);
  EngduinoLEDs.setLED(9, YELLOW);
  EngduinoLEDs.setLED(10, YELLOW);
  EngduinoLEDs.setLED(11, YELLOW);
  EngduinoLEDs.setLED(12, YELLOW);
  EngduinoLEDs.setLED(13, YELLOW);
  EngduinoLEDs.setLED(14, YELLOW);
  EngduinoLEDs.setLED(15, YELLOW);
```

So your final code should look something like this. (With your own tweaked timings in calls to the delay() function).

```
void loop()
{
  EngduinoLEDs.setAll(RED);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setLED(0, RED);
  EngduinoLEDs.setLED(1, RED);
  EngduinoLEDs.setLED(2, RED);
  EngduinoLEDs.setLED(3, RED);
  EngduinoLEDs.setLED(4, RED);
  EngduinoLEDs.setLED(5, RED);
  EngduinoLEDs.setLED(6, RED);
  EngduinoLEDs.setLED(7, RED);
  EngduinoLEDs.setLED(8, YELLOW);
  EngduinoLEDs.setLED(9, YELLOW);
  EngduinoLEDs.setLED(10, YELLOW);
  EngduinoLEDs.setLED(11, YELLOW);
  EngduinoLEDs.setLED(12, YELLOW);
  EngduinoLEDs.setLED(13, YELLOW);
  EngduinoLEDs.setLED(14, YELLOW);
  EngduinoLEDs.setLED(15, YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(GREEN);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
  EngduinoLEDs.setAll(YELLOW);
  delay(2000);
  EngduinoLEDs.setAll(OFF);
  delay(1000);
}
```

Now it's starting to look how we want it to, but it's the wrong way up. Now you can change the colours of the individual LEDs to red or yellow, you get the shape that you want. Try to make the program work in such a way that when the traffic light is Red/Yellow it looks something like this.

What next?

There are many things we can do next. If you're in a lab class, talk to one of the demonstrators, and they will give you another sheet that will tell you something about how you might make your code a bit more sophisticated, or how to work the button or one of the sensors. There's no substitute for experience – try things, and if they don't work, figure out why and fix them – there are many things you can do with the Engduino and the only rule is to be creative.

If you're at home or school working through this, you can always visit:

http://www.engduino.org/tutorials/

to find those sheets.

If you're a very advanced programmer, you can look at:

http://www.engduino.org/engduino_libraries_detailed/

And if you get stuck, by all means contact us on support@engduino.org