One Step Further

BLINK

By tradition, newcomers to Arduino and Arduino-based projects start by developing a blinking light. While Clyde is far from being a traditional lamp, in fact Clyde is anything *but* traditional, it's still a good place to start when one wants to learn how to change Clyde's behaviour. And in the blink of an eye, you will be adapting Clyde to your own needs!

There are two ways you can control Clyde programmatically, either through Clyde's firmware, or through Arduino's basic functionality. We will go through four different blinks, using Clyde's firmware, or using raw Arduino code, for both the task lamp and the ambient lamp.

1. Task Lamp Blink - firmware
2. Ambient Lamp Blink - firmware
3. Task Lamp Blink - Arduino
4. Ambient Lamp Blink - Arduino

Prerequisites

Before attempting these tutorials, you must at least go through Getting Started with Clyde - Software once. It will allow you to go back to Clyde's default behaviour, as we will be changing its behaviour in this tutorial. So if you want to go back to your out-of-the-box Clyde eventually - and you will get tired of Clyde's blinks after a while - we strongly suggest going through the steps once before attempting anything else.

Also, if you are new to Arduino programming, please familiarize yourself with this short document Clyde and Arduino which contains relevant links to the Arduino website.

Task Light Blink - Firmware

Let's roll our sleeves up, and get started!

First load the example into the Arduino IDE *(File / Examples / Clyde / Examples / ClydeBlinkWhiteTaskLight)*, and upload it to Clyde. The task light should start blinking.

This sketch is included in the downloaded files when you go through Getting started with Clyde - Software. If for some reason, it is missing for you, you can download it either from GitHub or from our friendly download file

DOWNLOAD ClydeSoftware.zip

The Task Light Blink sketch comes in three sections. At the top, you have all the necessary header files to access Clyde's firmware. You then have the **setup()** function where Clyde gets initialized. Two calls are necessary to initialize Clyde: **Wire.begin()** and **Clyde.begin()**. Finally, you have the third section is the **loop()** function where the logic required for blinking is to be found.

Let's spend some time in the loop() function, because that's where the fun happens. This is a fairly simple loop and the comments are clear. First we tell Clyde the value we want the task light to be at, and then we actually turn the light on. Then we wait a second (the delay function uses milliseconds). And we follow up with the opposite action. We tell Clyde what value the task light should be at, and we activate the task light and wait a second. Setting the light is a two step process: set the value, update the light. If you imagine Clyde as the genie in the lamp, this is the conversation you would be having:

> - Genie, I want you to prepare the lamp to be turned off
> *(genie prepares the lamp)*
> - Genie, on my command, turn the lamp off. NOW!
> *(genie turns the task lamp off)*
> - Genie, wait for my next command
> *(genie waits)* - Genie, prepare the lamp to be turned on
> *(genie prepares the lamp)*
> - Genie, on my command, turn the lamp on.... NOW!
> *(genie obeys)*
> - Genie, wait for my next command
> *(genie waits)*

Please note that there are versions of this code where the comments are wrong. The comments should read as here. As comments do not influence the sketch behaviour, don't panic if the comments in your code differ from what's here.

```
#include <Wire.h>
#include <EEPROM.h>
#include <Clyde.h>
#include <SerialCommand.h>
#include <SoftwareSerial.h>
#include <MPR121.h>

/*
 Clyde Blink White Task Light
 Turns on the white task light for one second, then off for one second, repeatedly.
*/

// the setup routine runs once when you press reset:
void setup() {
  // Initialize Clyde
  Wire.begin();
  Clyde.begin();
}

// the loop routine runs over and over again forever:
void loop() {

  //OFF
  Clyde.setWhite(255);  // setting the white light to 255 turns it OFF.
  Clyde.updateWhiteLight();
  delay(1000);  // wait for a second, or 1000 milliseconds

  //On
  Clyde.setWhite(0); // setting the white light to zero turns it ON.
  Clyde.updateWhiteLight();
  delay(1000);  // wait for a second, or 1000 milliseconds
}
```

Nota Bene:
There's something you need to be aware of. The Task Light behaves in reverse. Setting it to zero actually turns it on, and setting it to 255 turns it off. This is opposite to what is done in serial communication (see Speak Serial with Clyde) That's because the serial communication code swaps it around for you behind the scene while here, there's no hidden layer to swap it for you. So keep that in mind: for the task light (and not the ambient light), zero means on, and 255 means off.

Possible exercises:

1. Try and see if you can speed up the blink
   hint: change the 1000 milliseconds to a smaller delay, like 500 milliseconds. You need to change it in two places
2. Change the intensity of the blink, instead of just on/off, turn it on and half value.

Ambient Light Blink - Firmware

Now let's load the fun one - after all, having your desk illuminated for one second out of two isn't too conducive to concentration. But if we could vary the ambient light, now that would be cool! So load up the **ClydeBlinkAmbientRGBLight** sketch into the Arduino IDE and let's have a look!

The header files at the top are the same as for the task lamp, which is to be expected as those header files are the key to all of Clyde's functionality. The **setup()** function is also the same as previously seen. Only the **loop()** function differs.

Using three variables (r,g,b) for the ambient light's RGB components, the code determines that it will start with full red (r to 255, the other two set to zero). It then uses the **setAmbient()** method to set the internal values followed by **updateAmbientLight()** to trigger the actual light. This is similar logic to what was done with the task lamp, with the exception that here, 255 means full on, and zero means off.

> - Genie, prepare the ambient lamp to give us a pure red light
> *Genie prepares the lamp for red colour*
> - Genie, turn the ambient light on, now!
> *and there was light* etc...

Let's take a closer look at the **setAmbient()** method. It takes only one argument (the final colour), but needs three values (rgb). We get around this limitation by making a call to the RGB function which does exactly what we need : pass it three values corresponding to each of the R, G, B values and get back a colour which gets passed to setAmbient() in one gulp.

Have fun making changes to the sequence of colours, you are not limited to pure red, green and blue. Here are a few colours, but do experiment:

- aqua: red = 0; green = 255; blue = 255
- fuschia: red= 255; green = 0; blue = 255

```
#include <Wire.h>
#include <EEPROM.h>
#include <Clyde.h>
#include <SerialCommand.h>
#include <SoftwareSerial.h>
#include <MPR121.h>

/*
 Clyde Blink Ambient RGB Light
 Turns the ambient light red for one second, then off for one second
 then green and off and blue and off, repeatedly.
*/

// the setup routine runs once when you press reset:
void setup() {
  // Initialize Clyde
  Wire.begin();
  Clyde.begin();
}

// the loop routine runs over and over again forever:
void loop() {

  int r, g, b;
  //Red
  r=255, g=0, b=0;  // setting the r to 255, and g and b to 0 gives red.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

  //Off
  r=0, g=0, b=0; // setting the r, g and b to 255 turns it off.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

  //Green
  r=0, g=255, b=0;  // setting the g to 255, and r and b to 0 gives green.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

  //Off
  r=0, g=0, b=0; // setting the r, g and b to 0 turns it off.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

  //Blue
  r=0, g=0, b=255;// setting the b to 255, and r and g to 0 gives blue.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

  //Off
  r=0, g=0, b=0; // setting the r, g and b to 255 turns it off.
  Clyde.setAmbient(RGB(r, g, b));
  Clyde.updateAmbientLight();
  delay(1000);  // wait for a second

}
```

Task Light Blink - Arduino

Now let us compare this firmware approach to the raw Arduino approach. First, let us take a look at the basic Blink that Arduino recommends: http://www.arduino.cc/en/Tutorial/Blink. As you will be using Clyde's task lamp to blink, you do not need to have an external LED and you do not need to build the circuit found on the arduino tutorial. You already have all that you need with Clyde.

This is the basic Arduino Blink code, slightly modified for Clyde (the pin number was changed, and some comments). All explanations on the Arduino website are valid and apply equally well to Clyde.

Note the following differences and similarities between the code that uses Clyde's firmware and this new code:

- no header files are needed, as Clyde's firmware is not called.
- There's still a setup() function and a loop() function, just like before
- the code isn't as easily understood as the one that uses Clyde's firmware.
- while the arduino tutorial uses an LED on pin 13, Clyde has no use for an external LED but will address its task lamp by using pin 11.
- the final size of the compiled code is smaller than it was with the firmware, but that is mainly due to the fact this is very simple interaction. With complexity, the size may not always be smaller.

This approach takes us closer to the actual hardware. We are required to know about pin layouts, and high/low voltage. It is not as easily accessible as the firmware approach which we saw first. Our Genie in a Lamp isn't as bright as before. This time we have to spell out everything:

> - Genie, put some voltage onto pin 11
> - Genie, wait
> - Genie, remove the voltage from pin 11
> - Genie, wait

You might notice there is no easy way to control the intensity of the lamp with this new approach. Although Arduino does let us play with the intensity, we do have to use a different set of functions. If you're interested in exploring this, you can take a look at their fade tutorial. Remember to use pin 11 for the task lamp.

```
 /*
  Blink
  Turns on Clyde’s task lamp on for one second, then off for one second, repeatedly.
  This example code is in the public domain, and comes from
  http://arduino.cc/en/Tutorial/Blink
*/

// Pin 11 is the pin associated with Clyde’s Task Lamp
// give it a name:
int led = 11;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);

}

// the loop routine runs over and over again forever:
void loop() {

// remember: TASK LAMP is inversed
  digitalWrite(led, HIGH);   // turn the TASK LAMP OFF (HIGH is the voltage level)
  delay(1000);               // wait for a second or 1000 milliseconds
  digitalWrite(led, LOW);    // turn the TASK LAMP ON by making the voltage LOW
  delay(1000);               // wait for a second or 1000 milliseconds
}
```

Ambient Light Blink - Arduino

The ambient lamp needs three pins to control each of its RGB component independently of each other. Red will be controlled via pin 5, Green via pin 6 and Blue via pin 9.We will follow the same logic as we did for the task lamp through Arduino, except that we are spread over three pins this time, so we will need three times the code. It's becoming evident that the Arduino approach can get cumbersome pretty quickly.

```
/*
declare appropriate pins for RGB
*/

const int rLED = 5;    // pin that controls the RED component
const int gLED = 6;   // pin that controls the GREEN component
const int bLED = 9;   // pin that controls the BLUE component

void setup() {
  pinMode(rLED, OUTPUT);  // declare our red pin as output
  pinMode(gLED, OUTPUT); // declare our GREEN pin as output
  pinMode(bLED, OUTPUT); // declare our BLUE pin as output
}

void loop() {
```

```
// set ambient light to RED
  digitalWrite(rLED, HIGH);   // all red
  digitalWrite(gLED, LOW);  // no green
  digitalWrite(bLED, LOW);  // no blue
  delay(1000);   // wait 1000 milliseconds

// turn ambient light OFF
  digitalWrite(rLED, LOW);   // no red
  digitalWrite(gLED, LOW);  // no green
  digitalWrite(bLED, LOW); // no blue
  delay(1000);                    // wait 1000 milliseconds

// turn ambient light to GREEN
  digitalWrite(rLED, LOW);   // no red
  digitalWrite(gLED, HIGH); // all green
  digitalWrite(bLED, LOW);  // no blue
  delay(1000);                     // wait 1000 milliseconds

// turn ambient light OFF
  digitalWrite(rLED, LOW);   // no red
  digitalWrite(gLED, LOW);  // no green
  digitalWrite(bLED, LOW); // no blue
  delay(1000);                    // wait 1000 milliseconds

// turn ambient light to BLUE
  digitalWrite(rLED, LOW);  // no red
  digitalWrite(gLED, LOW);  // no green
  digitalWrite(bLED, HIGH);  // all blue
  delay(1000);                     // wait 1000 milliseconds

// turn ambient light OFF
  digitalWrite(rLED, LOW);   // no red
  digitalWrite(gLED, LOW);  // no green
  digitalWrite(bLED, LOW); // no blue
  delay(1000);                    // wait 1000 milliseconds
}
```