

# ARDUINO IO SIMULATOR DRAG & DRAW

## USER MANUAL



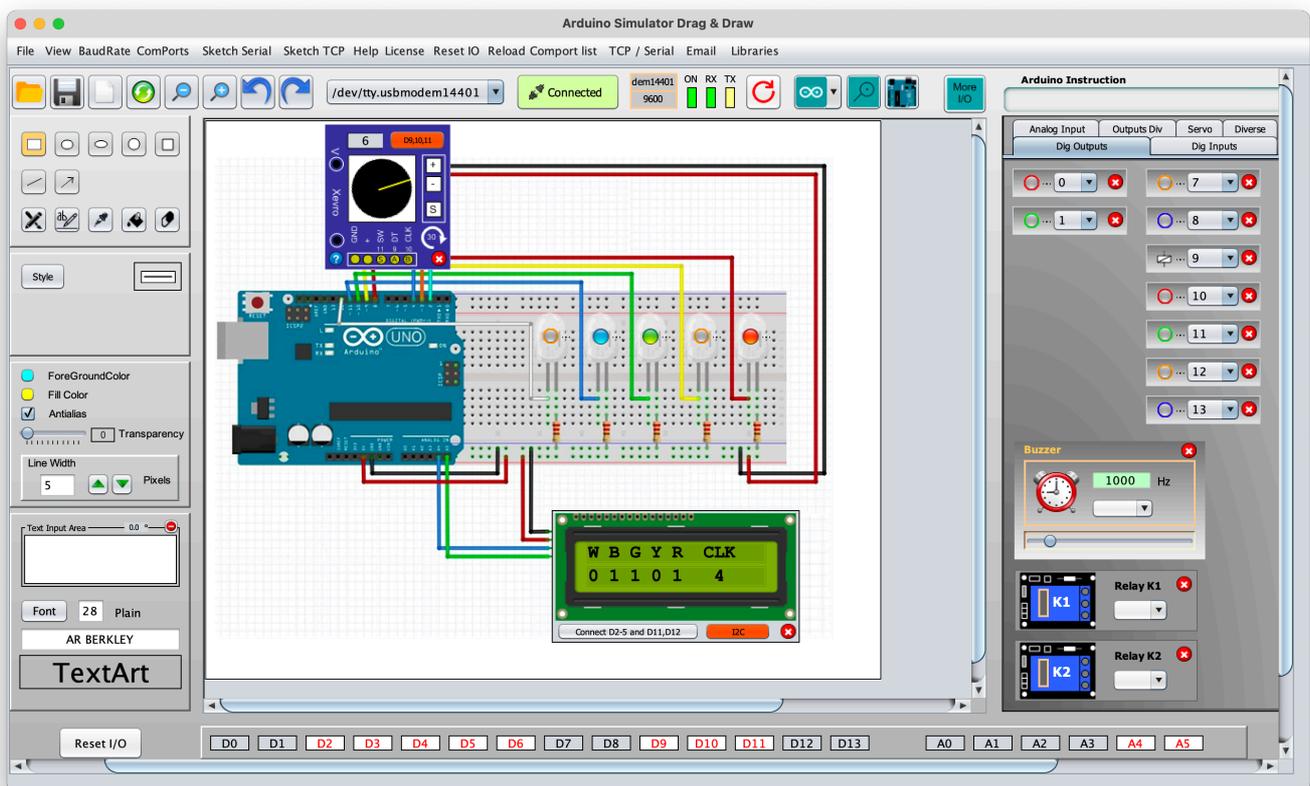
### XEVRO

Version 1.5.4

Xevro© 2022

Louis D'Hont & Marc Van Den Berge

This manual describes all features and capabilities of the Arduino IO Simulator.



# Table of Contents

|  |    |
|--|----|
| Arduino IO Simulator Drag & Draw ..... | 1  |
| Table of Contents .....                | 2  |
| Arduino IO Simulator .....             | 4  |
| Arduino IDE.....                       | 4  |
| License Registration.....              | 5  |
| Drawing shapes.....                    | 6  |
| Style.....                             | 6  |
| Drawing tools.....                     | 7  |
| Toolbar .....                          | 8  |
| Drag & drop .....                      | 13 |
| Extra IO .....                         | 14 |
| Arduino Board Viewer.....              | 15 |
| Use the digital IO .....               | 17 |
| Leds.....                              | 17 |
| Buzzer.....                            | 17 |
| Relay.....                             | 17 |
| Buttons .....                          | 18 |
| Squarewave.....                        | 18 |
| Motion detection .....                 | 18 |
| DHT11 sensor .....                     | 19 |
| Noise detection.....                   | 19 |
| Sliders.....                           | 19 |
| 7 segment display .....                | 20 |
| Tone Melody.....                       | 20 |
| Bargraph .....                         | 20 |
| 8 Digit 7 segment display.....         | 21 |
| Sound generator .....                  | 24 |
| Analog input box.....                  | 24 |
| PWM value box.....                     | 24 |
| Servo .....                            | 25 |
| Motor wheels .....                     | 26 |
| Barrier.....                           | 28 |
| Keypad.....                            | 29 |
| LCD display.....                       | 29 |
| RGB led.....                           | 30 |
| RGB Slider .....                       | 30 |

|  |    |
|--|----|
| Hall effect sensor .....                                       | 31 |
| Rotary encoder .....   | 35 |
| Ir remote control.....   | 36 |
| Use the IR remote library .....                                | 39 |
| Stepper motor 28-BYJ48.....                                    | 42 |
| Input/feedback .....   | 42 |
| Stepper library .....  | 43 |
| Calculate the Steps per Revolution for the stepper motor ..... | 44 |
| MotorControl .....   | 45 |
| MAX7219 LED dot matrix display .....                           | 46 |
| About the MAX7219 LED driver .....                             | 46 |
| How to connect the dot matrix display to the Arduino.....      | 46 |
| Hardware SPI pin locations Arduino Uno.....                    | 47 |
| Hardware configuration in Arduino .....                        | 51 |
| How the code works.....  | 51 |
| Other text effects.....  | 56 |
| Make your own sprites .....                                    | 57 |
| Arduino TCP ETH-WiFi connection.....                           | 58 |
| TCP connection button .....                                    | 59 |
| Reset Arduino Button .....                                     | 60 |
| New Sketch .....   | 61 |
| Warnings and Errors .....                                      | 63 |
| TCP connection on a Wide Area Network .....                    | 64 |
| Wireless connection with Arduino Wifi .....                    | 65 |
| Port forward on modem B-box3 .....                             | 66 |
| Arduino Simulator sketch.....                                  | 67 |
| Configure the serial port.....                                 | 68 |
| Save and Restore of settings .....                             | 69 |
| Serial Monitor .....   | 70 |
| Email Logger .....   | 71 |
| The email box.....   | 73 |
| Check for updates.....   | 75 |
| Sources.....   | 76 |

## ARDUINO IO SIMULATOR

The Arduino IO Simulator gives you the tools and components you need to simulate your Arduino IO. It's made for quick tests and small projects, there is still further developed in order to obtain the widest possible IO functions.

This Arduino IO Simulator is designed to test an Arduino program quickly with the Arduino board without really having connections to external IO (buttons, potentiometers, LEDs, LCD displays, ...). Add a nice custom drawing around it to get a better simulation experience.

To use the simulator, we need 3 programs:

- Java JDK
- The Arduino simulation program
- The Arduino IDEs

In order to use the simulator, we need to download the Java JDK on our computer, you can find the download link on the website of Xevro or go to Java where you can find the latest version.

## ARDUINO IDE

For we start using the Arduino IO Simulator we need the Arduino software, it is also free available on the Arduino website: <http://arduino.cc/en/Main/Software>



### Download the Arduino Software

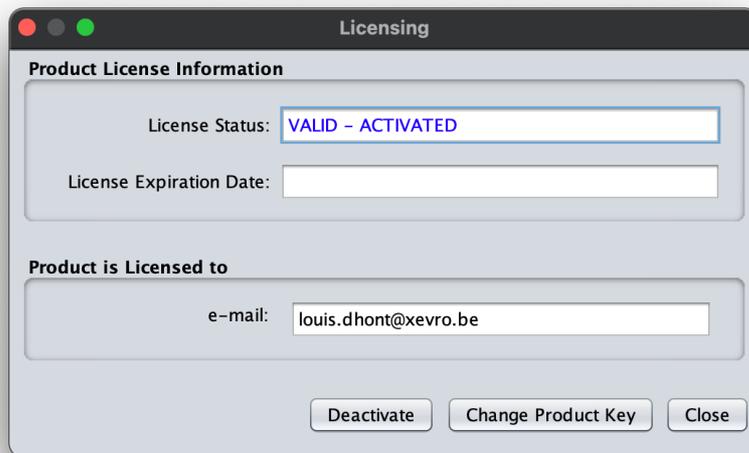


## LICENSE REGISTRATION

The Simulator is available in a full 30-day trial and a paid version. The first time you open the program there will be a license activation screen popping up where you can add the license key and activate the 30-day trial or the paid license.

Click on the 'Change Product Key' to insert the license key you copied on the website, after entering this you need to click on 'activate'.

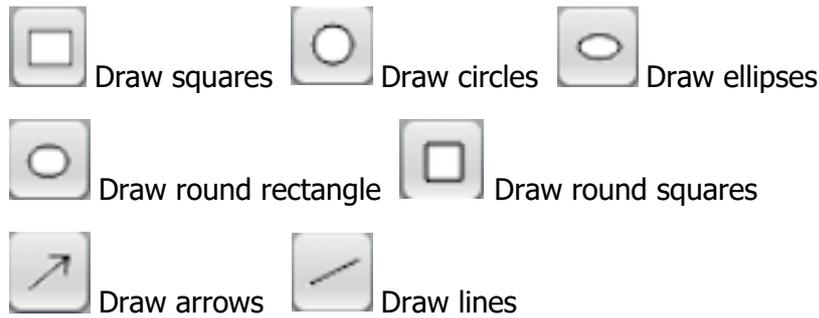
When you want to change the trial as a paid version just replace the license key by clicking on the 'Change Product Key'. After entering the license key, you can activate the key with the **'activate'** button.



### Change license key

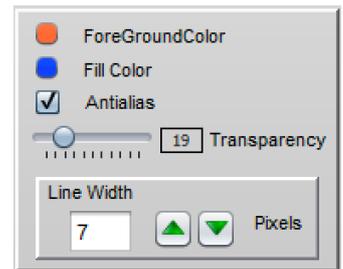


## DRAWING SHAPES



The first 4 tools have the same color, these colors are stored for the next use. The line and arrow tools each have a different color that is customizable. The transparency is customizable with the slider.

The line thickness of all tools can be adjusted.



## STYLE

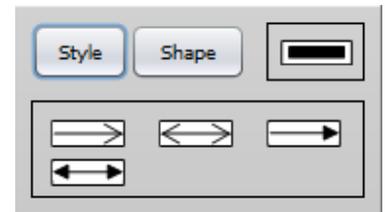
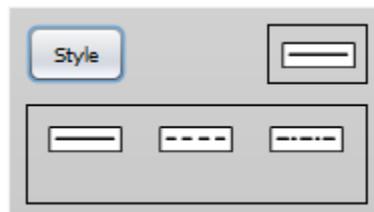
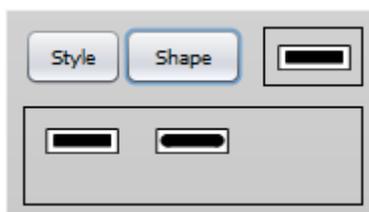
If a tool is selected, you can change the line using the style button.

There is choice of a full line , dotted line  or long dotted line .

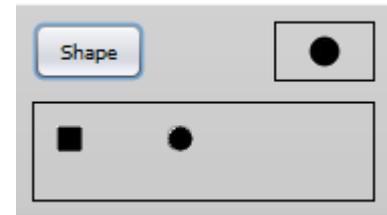
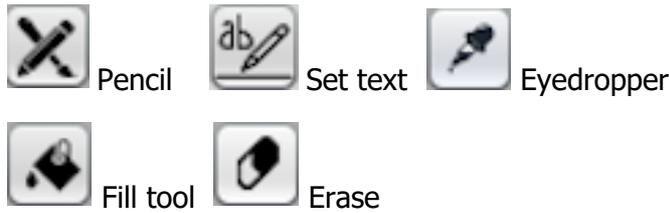
If you draw a line, you can customize the style and shape. The style for the line is the same as above. With the shape you can customize the beginning and end point, with the arrow just the beginning. Press the shift key to draw a perfect horizontal or vertical line.

There is a choice of a right angle  or a rounded corner .

The arrow point can be adjusted, there is a single arrow , each side an arrow , a fat arrow  and a fat arrow at both ends .



## DRAWING TOOLS



With the pencil you can draw free lines. The shape of the beginning and end of pencil can be changed, there is a choice of a square ■ or a circle ●. Press the 'shift' key to make the line straight.

To put text in the field, type the text in the text-field. The font can be customized by pressing the 'font' button. With the small red button, you can clear the text-field. There are 3 lines in the 'Text Area' available.

If you place the text in the field while holding the left mouse button pressed, then press '+', '-' key and the text begins to rotate, the number of degrees can be seen next to the red clear button.

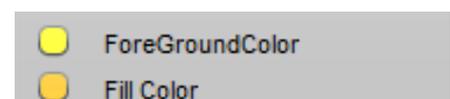
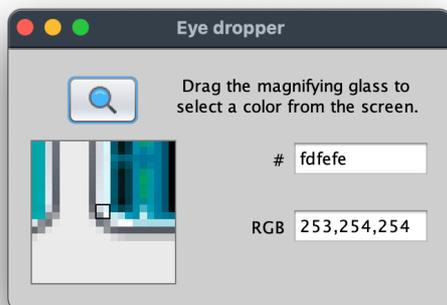


If you see a color that you would like to use, use the eyedropper to find the color you want. To activate the eyedropper, click on the 'eye' button and hold down the mouse while moving, then move across the screen to scan the colors.

With the fill tool it is possible to fill a level in the form of one of the 5 shapes. Select one of the 5 shapes and select the fill tool and draw.

Select the erase tool to remove. Adjust the thickness by changing the value in the text field or by increasing or decreasing the value with the arrows.

To change the color of the foreground and fill color, click on the colored buttons, this opens a color chooser panel.



## TOOLBAR

The zoom in  and out  buttons are used to zoom in or out on the work area. When the  minimum or  maximum zoom factor is reached, the button will turn yellow. If you have a slow processor, the drawing and zooming tools will maybe work slower.

The undo  and redo  buttons are used to undo or redo the things you changed on the picture.

Save the picture to a specific place.

To change the background, press the folder button, this opens a file chooser where you can choose an image.

Add a blank white sheet to get the drawings cleaned up.

The refresh button is used to repaint the worksheet if there is a graphical issue.

Open a new Arduino IO Simulator sketch (sketch can't be saved, use save as...)

Use the reset button to reset the Arduino.

Select an option in the dropdown combobox to open a new serial, ETH or WIFI sketch.

Click this button to open the serial monitor. By clicking it again, you close it.

By clicking on the Arduino board icon, you will get an Arduino board viewer that shows you all the connected IO pins. The window is made a top-level window to view it all the time while working in the simulator.

To get more components click on the 'more I/O' button. By clicking it again, you close it.



# HOW TO USE IT

The Arduino IO Simulator is very easy to understand and work with. The Simulator requires 5 simple steps to simulate a project.

- 1. Connect the Arduino board**
- 2. Upload your custom Arduino code with the corresponding library file**
- 3. Add the used libraries**
- 4. Select the used in-outputs in the Arduino IO Simulator**
- 5. Connect the Arduino IO Simulator to the Arduino board with the right serial port**

## 1. Connect the Arduino Board

The Arduino IO Simulator works with a lot of Arduino boards:

- Arduino UNO
- Arduino Mega
- Arduino Leonardo
- Arduino UNO WIFI Rev2 (TCP connection)
- Arduino UNO with Ethernet shield W5100 (TCP connection)
- Arduino...

**Attention:** Only the digital and analog pins that are available on the Simulator can be used! Disconnect the Arduino IO Simulator before uploading the Arduino code.

## 2. Upload your custom Arduino code with the corresponding library file

Open the simulator and go to 'Help -> Arduino UNO programming code -> Arduino UNO programming code (ino)'.

This will open a Arduino (ino) file with the corresponding library and important code in it.

## 3. Add the used libraries

In order to let the Simulator, understand the code, we have created our own libraries. To maintain the usability, we have decided to keep the instructions as they are but we changed the libraries a bit so they are compatible with our software.

There are a few libraries available to use. The simulator program library is necessary for the digitalWrite... instructions. To use the 16x2 LCD display you must add our liquidCrystalSim library in order to use it with the simulator. All the instructions are the same.

## 4. Select the used in-outputs in the Arduino IO Simulator

Each input and output on the Simulator have a selection box where the used digital or analog pin can be connected.

## 5. Connect the Arduino IO Simulator to the Arduino board with the right serial port

The Arduino IO Simulator knows which port is the Arduino board.

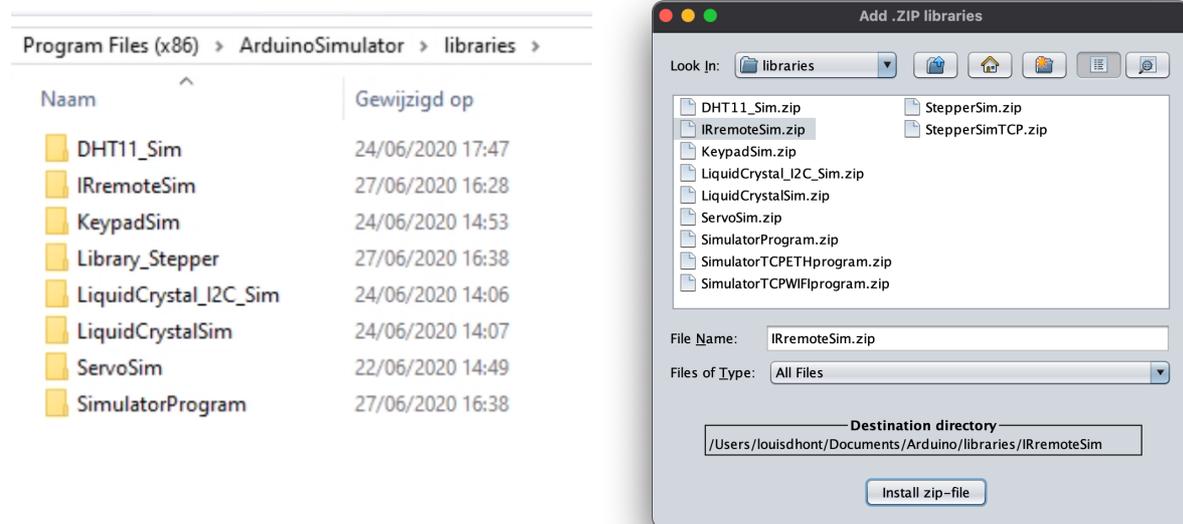
Make sure the Arduino is disconnected while uploading the Arduino code.

## CODE CHANGES

The Arduino IDE works with instructions that the IO read and write, by adding the libraries to your project it is possible to simulate the project.

We don't want to change the real instructions, we decided to edit the libraries, so they are compatible with our software. The core SimulatorProgram library will be added to the Arduino/libraries folder in the documents folder the very first time you open the simulator as well as all the other libraries (this is present in version 1.5.1 and higher). In the Simulator software is a function available to add a library to add more libraries to the Arduino IDE.

### All available Simulator libraries



Go inside a folder and select the .zip file, click on the 'install zip-file' button to install the library. The library will be automatically recognized by the Arduino IDE. by adding the Arduino include statement to your project the library will be used in your project, and you can make simulations with the Arduino board.

Use the SerialInput("x") to read a value out of the serial communication line.

```
if (SerialInput.equalsIgnoreCase("START")){
    digitalWrite(relais,HIGH);
}
```

**The stepper motor commands are the same as the ones in the stepper library provided by Arduino.**

```
stepper.setSpeed(value between 0-200);
Stepper(STEPS, pin1, pin2);           For 2 coil winding
Stepper(STEPS, pin1, pin2, pin3, pin4);   For 4 coil winding
setSpeed(rpm);
step(STEPS);
```

More information about the stepper commands:  
<https://www.arduino.cc/en/Reference/Stepper>

**attention:**

Remember to adjust the 'Void Setup to initialize the real IO!

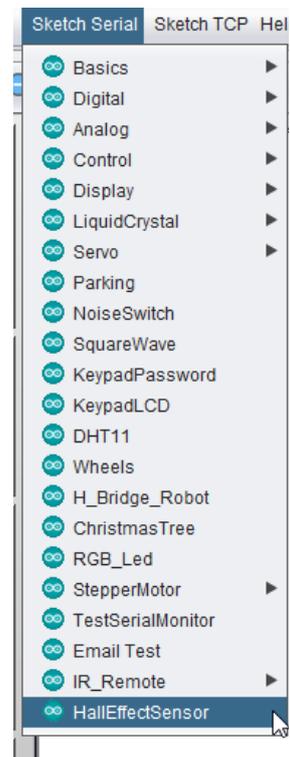
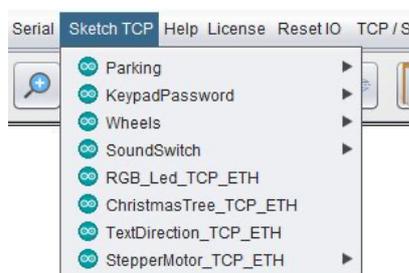
```
Example:   pinMode (0, OUTPUT);
           pinMode (1, INPUT);
```

In each sketch are the instructions changed, so you only need to connect the inputs and outputs through the screen and the sketch should work.

**How to test an example:**

1. Open an example sketch and upload it into the Arduino board
2. Add the used components (I/O) in the worksheet. (There are some examples that are already saved in the restore settings)
3. Connect the Arduino IO Simulator with the board
4. Simulate your Arduino in and outputs on the simulator

There are 8 sketches for TCP communication. The first 4 examples have also been made to work with a wireless Arduino WIFI connection.



## Check your program with 'CheckVar' variables

With Checkvars it is possible to check your sketch, you can insert a variable on a line in your sketch and via the serial monitor you can follow the values of the variables.

You can enter many different variables as:

Int, long int, long unsigned int, word, double, float, char, string, Boolean

### Instruction:

CheckVar(num , var);

num: integer from 0 to 32768

var: Int, long int, long unsigned int, word, double, float, char, string, Boolean

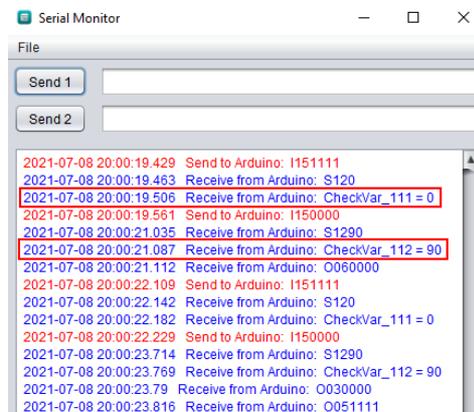
### Example sketch parking:

I want to check the variables 'BarUp' and 'BarLow' in the sketch, and I give the CheckVars the numbers 111 and 112 for BarUp and BarLow.

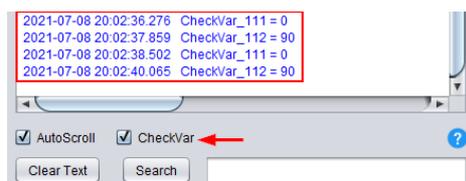
In the serial monitor you can follow the variables CheckVar\_111 and CheckVar\_112 and their values.

```

if (digitalRead(Exit) == 1)
{
  if (Available != CAPACITY) {
    Available++;
    myservo.write(BarUp);
    CheckVar(111,BarUp);
    delay(1500);
    myservo.write(BarLow);
    CheckVar(112,BarLow);
  }
}
    
```



It is also possible to see only the CheckVars by check the box 'CheckVar'.



## DRAG & DROP

All the IO components can be dragged into the worksheet. The components can be resized by clicking on the component image itself or on top of it.

For example:

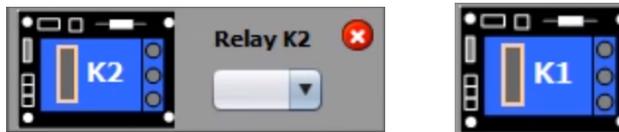
Click on the '...' by the LEDS to hide the IO selection.



Buttons:



The relay can be resized by clicking on the image itself.



The slider can be resized by clicking in the green area.



The bargraph can be resized by clicking on the bargraph itself.



Some components like the 7-segment display, LCD display keypad can't be resized. With the 'Reset IO' button you can set the components back to its original position. The button is located in the left corner below, on the top menu bar or in the Arduino board IO viewer.



## Drag functionally

When you drag the component in the worksheet you can see which instruction the component can be controlled with.

For example, when you choose a button, it shows you DigitalRead().



A LED will show DigitalWrite(), that's the instruction that will control the LEDs.

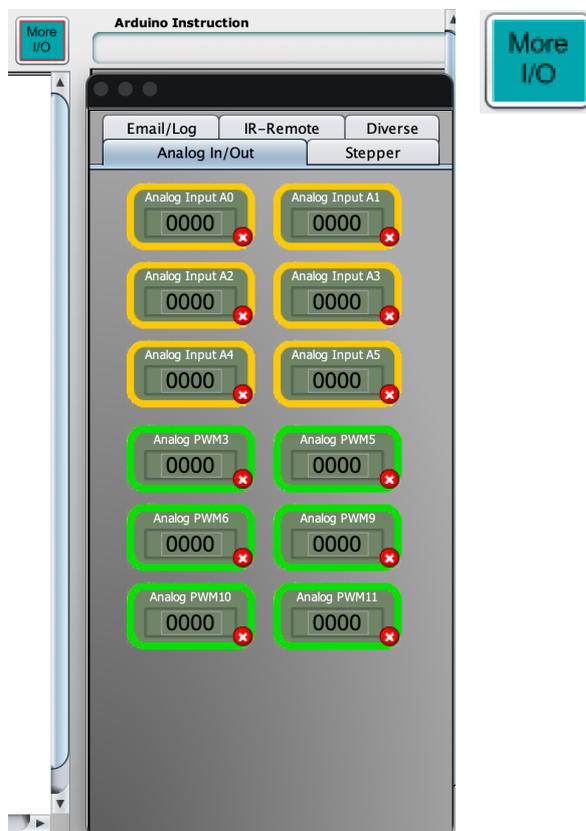


The component border will turn red when it's ready to drag into the worksheet.



## EXTRA IO

In order to add more IO parts, we have decided to provide a button that opens a window with more parts in it. Click on it again to hide the window.



## ARDUINO BOARD VIEWER

By clicking on the Arduino board button, a window will open in which you will see an Arduino UNO board with all the used IO pins on it. The Reset IO button will clear all the IO pins to start with selecting IO again. The RX/TX LEDs are also visually simulated.

The Arduino UNO board window is set as a top-level window to make sure you can always see the Arduino board while you are simulating projects.

The real IO connect method in the setup will allow the simulator library to use simulated IO together with real IO pins if you want to use connected components on . When you perform a hardware reset on the real Arduino board or a reset from the simulator Arduino board you will lose all real IO connections, to get these connections back you must enter the following instruction in the setup.

When working with TCP you must place the instruction 'RealIO\_Connect()' after the delay(5000).

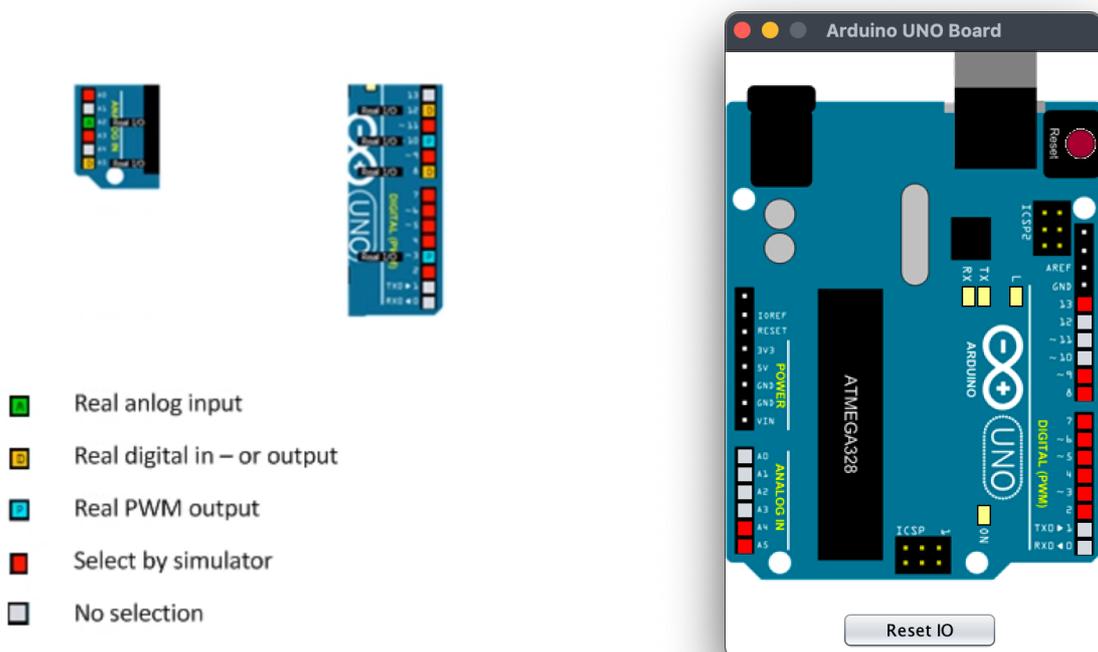
```

void setup() {
  //***** code for SimulatorTCP *****
  Serial.begin(9600); // Simulator Serial Con
  inString.reserve(25);

  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
  }
  Ethernet.begin(mac, ip, gateway, subnet);
  server.begin(); // Start the server.
  Serial.print("Init");
  delay(5000);
  RealIO_Connect(); // Used for connection between arduino and real IO , place
  //lcd.port(portname); // The lcd library needs this portname
  //myservo.port(portname); // The servo library needs this portname
  //stepper.port(portname); // The stepper library needs this portname
  //irrecv.port(portname); // The IRremote library needs this portname
  //irsend.port(portname); // The IRremote library needs this portname
  //***** end code SimulatorTCP *****
}

void setup() {
  // Simulator Serial Connection code
  Serial.begin(9600);
  inString.reserve(10);
  RealIO_Connect(); // Used for connection between
  
```

When you click on a pin you can change the status of the pin. To use an analog or digital pin as a real in or output you will have to click ones and it will show a 'D' OR 'A' in it. A PWM pin will show 'P'. Each time you click on a pin, the status will be sent to the Simulator library on the Arduino board.



**Attention when working with Ethernet shield W5100:**

The Ethernet Shield use SPI (Pin 11, Pin 12, and Pin 13) along with Pin 10 for the Ethernet SPI Slave Select and Pin 4 for the SD Card SPI Slave Select. It shouldn't be using pin 3. It can optionally use Pin 2 for the W5100 "INT" signal, but the Ethernet library doesn't support it. It shouldn't be using Pin 14 (A0) or 15 (A1).

The FreqCount library ([http://www.pjrc.com/teensy/td\\_libs\\_FreqCount.html](http://www.pjrc.com/teensy/td_libs_FreqCount.html)) says:

**Arduino Uno:** Frequency Input Pin: 5 Pins Unusable with analogWrite(): 3, 9, 10, 11

That explains why 3, 9, 10, and 11 won't work for PWM.

So that leaves you with PWM pins:

3: Disabled by FreqCount

5: Needed buy FreqCount for counter input

6: Available

9: Disabled by FreqCount

10: Disabled by FreqCount and used by Ethernet Shield

11: Disabled by FreqCount and used by Ethernet Shield

If you use the Wire library (`#include <Wire.h>`) then you cannot used A4 and A5

Board            I2C / TWI pins

Uno, Ethernet    A4 (SDA), A5 (SCL)

## USE THE DIGITAL IO

All the IO components are placed in a tab pane that can be changed by clicking on the tabs above. The button 'More I/O' will open a window with more IO available.



## LEDS

There are 14 LEDs available, for every pin of the Arduino 1 LED. LED D4 to D9 also have the option of choosing an analog pin as a digital pin (D14-19). D14 for the corresponding number is A0, example: D4 gets D14 in the combo box, D6 gets D16 and so on...

Use the combobox to connect it with the Arduino. The cross button is used to set the LED back to its original place in the tab pane. By clicking on the LED you can change the color.

The LEDs can be controlled with the `digitalWrite()` function.



## BUZZER

The buzzer is used to make a noise with a custom frequency. The combobox is used to connect the buzzer with the Arduino. D14-19 are the analog pins represented in a digital number. The cross button is used to set the buzzer back to its original place in the tab pane. Click on the clock to resize the buzzer and hide the settings.

The buzzer can be controlled with the `digitalWrite()`; function. By sending out a `digitalWrite(pin, HIGH)`; signal in the Arduino code, the buzzer will make a noise with the adjustable frequency.



## RELAY

There are 2 relays to simulate a relay contact. The combobox is used to connect the relay to one of the 14 IO pins. D14-19 are the analog pins represented in a digital number. The cross button is used to set the relay back to its original place in the tab pane. Click on the relay itself to resize and hide the combobox settings.

The relay can be controlled with the `digitalWrite()` function.

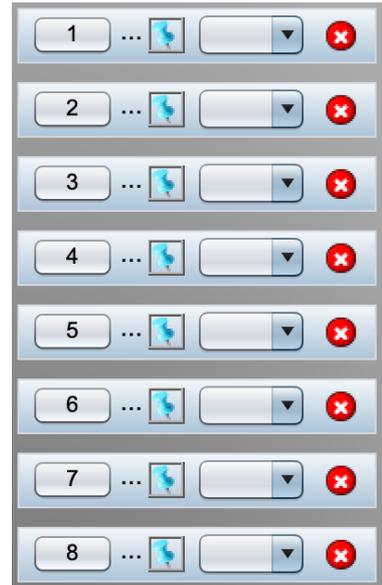


## BUTTONS

There are 8 buttons available. The combobox is used to connect the button to one of the 14 IO pins. D14-19 are the analog pins represented in a digital number.

The cross button is used to set the button back to its original place in the tab pane. Click on the `...` to resize and hide the combobox settings. The light blue pin can be used to hold down the button while doing other things, the border changes to red when it's pressed.

The buttons can be controlled with the `digitalRead()` function.

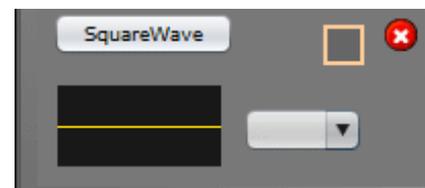


## SQUAREWAVE

The squarewave sends pulls signals to the Arduino, when the signal is high you see the grey square lights up `red`. The combobox is used to connect the square wave to one of the 6 analog pins (A0-A5 = D14-19).

The cross button is used to set the square wave back to its original place in the tab pane. When you click on the `SquareWave` button there opens a second window with a slider to change the frequency.

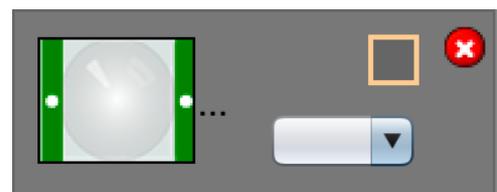
The square wave can be controlled with the `digitalRead()` function.



## MOTION DETECTION

The motion sensor sends signals to the Arduino every time there is a movement (click on the sensor image), when the signal is high you see the gray square lights up red. The combobox is used to connect the motion sensor to one of the 14 digital pins. D14-19 are the analog pins represented in a digital number. Use the `...` to resize and hide the settings. The cross button is used to set the sensor back to its original place in the tab pane.

The motion sensor can be controlled with the `digitalRead()` function.



## DHT11 SENSOR

The DHT11 simulates the temperature and humidity. The combobox is used to connect the sensor to one of the digital pins (D2-13). D14-19 are the analog pins represented in a digital number. The cross button is used to set the sensor back to its original place in the tab pane. The 2 sliders are used to change the value of the temperature (°C) and humidity (%).

The DHT sensor can be controlled with the `DHT.read11(DHT_11, Sim_Tmp, Sim_Hum)` function. Use the DHT11 example to test this sensor. The `Sim_Tmp` and `Sim_Hum` is used by the simulator and are the 2 sliders for the °C and % that need to be given to the library by passing it through with the function `read11()`.

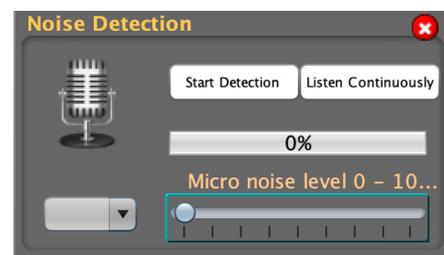


## NOISE DETECTION

The noise detection is used to send an analog (0-1023) signal to the Arduino depends on the noise level. The combobox is used to connect the noise detector to one of the 6 analog pins (A0-A5).

The cross button is used to set the noise detector back to its original place in the tab pane. When you click on the 'Start Noise Detection' the detection starts listening to the microphone noise level. When the noise level exceeded the slider value then it will send the signal (0-1023) to the Arduino. The limit value in the Arduino code needs to be lower than the noise detection slider because the signal will be sent when the noise is detected. With the 'Listen Continuously' button you can let the simulator simulate a microphone that listens continuously until it reaches the threshold value set with the slider.

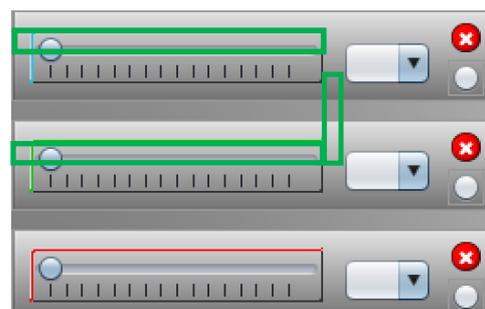
The noise detection can be controlled with the `analogRead()` function.



## SLIDERS

There are 3 sliders to connect with one of the 6 analog pins (A0-A5). The cross button is used to set the slider back to its original place in the tab pane. The green rectangle shows the place you can click on to hide the combobox and make the component smaller. The value is sent to the Arduino after releasing the mouse when changing the slider, if you click on the small circle the Simulator will send constant values to the Arduino.

The sliders can be read by the Arduino with the `analogRead()` function.

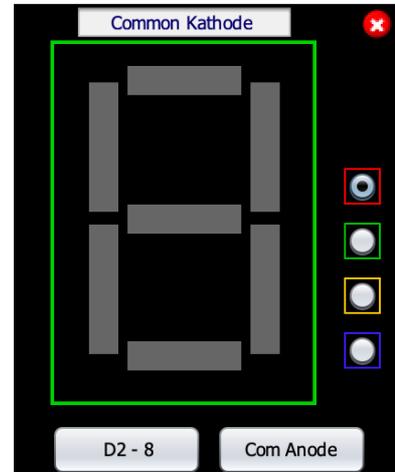


## 7 SEGMENT DISPLAY

The 7-segment display has 7 digital pins that can connect to D2-8 on the Arduino. The display can be connected in common anode or common cathode. The cross button is used to set the display back to its original place in the tab pane.

To light up the display-use `digitalWrite(D2-8);`

See the example: Parking



## TONE MELODY

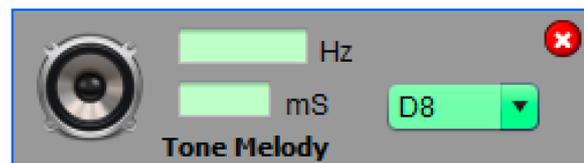
The tone melody can be connected to a digital pin from D0-9 of the Arduino. The cross button is used to set the tone melody back to its original place in the tab pane. The frequency and time of the sound (milliseconds) are present in the light green boxes.

Use `tone(pin, f, d);` and `noTone(pin);` (See example: Tone Melody)

`noTone()` stops playing sound.

**f** = frequency

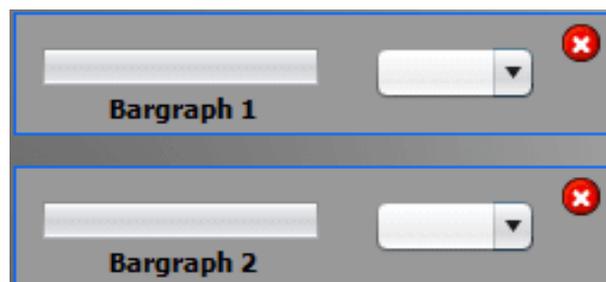
**d** = duration



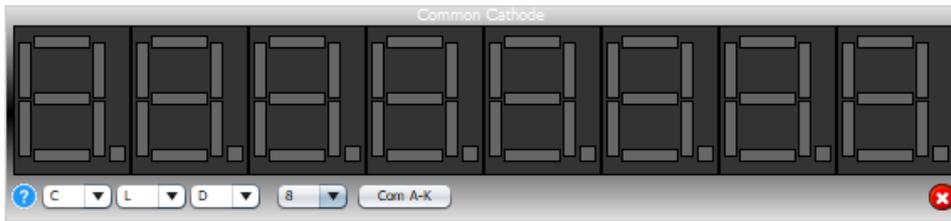
## BARGRAPH

The bargraph can be connected to one of the 6 digital PWM pins of the Arduino. The cross button is used to set the bargraph back to its original place in the tab pane. The bargraph shows the % of your value (0-1023), this can be used to simulate a PWM signal as a % bar. By clicking on the bargraph itself you can resize it and hide the combobox to make it smaller.

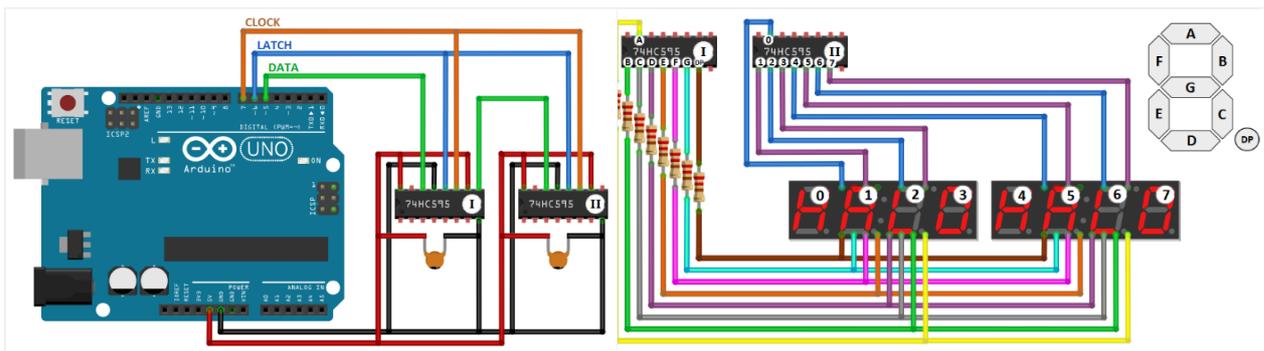
Use `analogWrite(pin, value);` to control the bargraph (See example: sound switch, use the restore function to get the IO already dragged into the work sheet).



## 8 DIGIT 7 SEGMENT DISPLAY



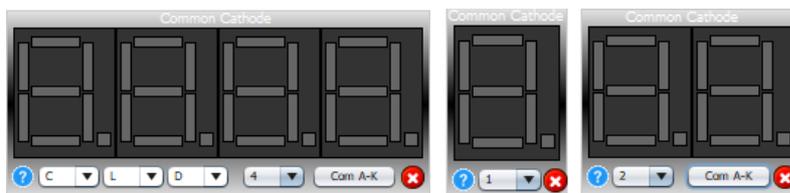
This is an 8-digit 7 segment display that uses the shift registers 74HC595. This means that we only need 3 pins of the Arduino board to drive the display.



There are 3 combo boxes L (latch), C (clock), D (data) that you can connect to the digital pins (D2 to D13) of the Arduino board.

With the button 'Com A-K' you can choose to work with common anode or common cathode.

With the combobox '1-8' you can set the display size from 1 digit to 8 digits.



A library is needed to control the display from the Arduino board. We have chosen the Miguel Pynto library "**ShiftDisplay-3.6.1**" because it is easy to use.

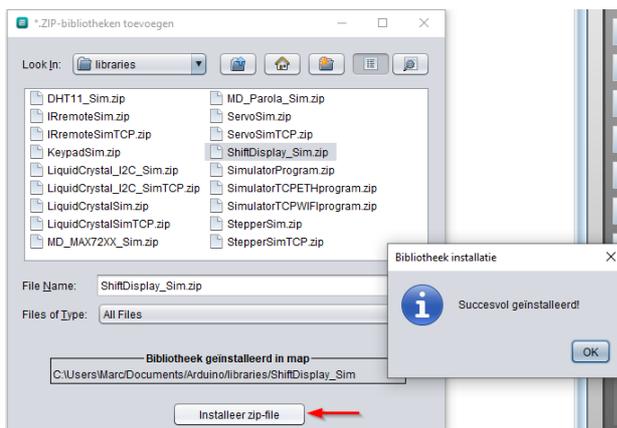
We modified the library to communicate with the simulator and named it "**ShiftDisplay\_Sim**".

## Features of the library:

Arduino library "**ShiftDisplay\_Sim**" for controlling 7-segment displays using 74HC595 shift registers (by MiguelPynto)

- Displaying digits and text
- Merge multiple displays into one for up to 8 digits
- Compatible system with common cathode and common anode
- Only 3 pins used on the Arduino board

Normally the library "**ShiftDisplay\_Sim**" is installed automatically when the simulator is installed, but this can also be done manually:



The instructions accompanying the library can be found on Miguel Pynto's website with text and explanation:

<https://miguelpynto.github.io/ShiftDisplay>

In the sketches we use the same instructions as the original library '**ShiftDisplay-3.6.1**', we have modified the library so that it sends its data to the simulator via serial communication.

It is important to note that the **<ShiftDisplay\_Sim.h>** library must always be above the simulator library **<SimulatorProgram.h>** otherwise this will not work.

```
#include <ShiftDisplay_Sim.h>
#include <SimulatorProgram.h>
```

**Remark:**

Because the simulator itself uses serial communication, it is not yet possible for the display to work with the library `'#include <Wire.h>'`.

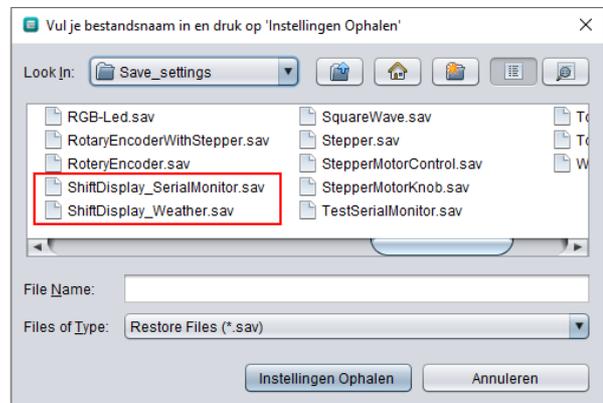
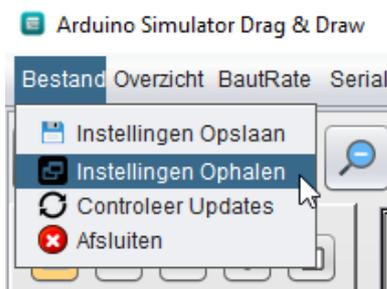
Do not forget to connect the comboboxes "C, L, D" to the Arduino board otherwise the display will not work.

**Example of serial sketches in the simulator:**



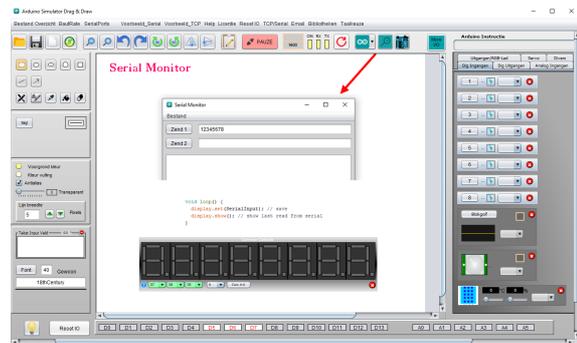
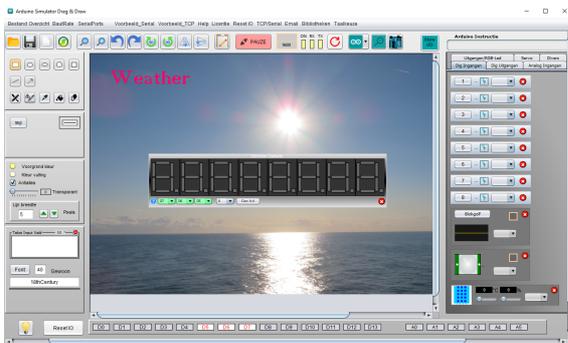
Click on 'Weather' or 'Serialmonitor', the Arduino IDE starts up with the sketch.

Retrieve simulator settings for the sketches:



Weather

Serial Monitor

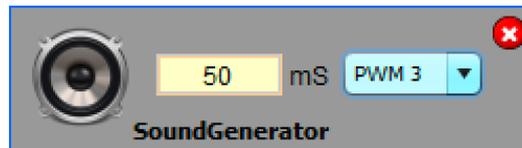


## SOUND GENERATOR

The sound generator can be connected to one of the 6 digital PWM pins of the Arduino. The cross button is used to set the sound generator back to its original place in the tab pane.

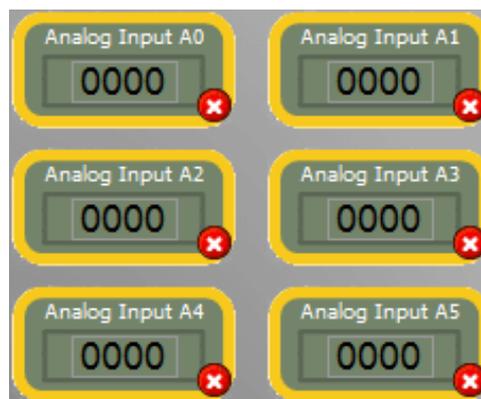
By clicking on the sound box image, you can resize it and hide the combobox to make it smaller. By changing the time (ms) you change the duration that the sound goes off (1ms – 10 000ms). The frequency can go from 10hz to 10Khz.

Use `analogWrite(pin, value);` to control the sound generator.



## ANALOG INPUT BOX

The analog input box shows the value of each analog pin (A0-A5). The cross button is used to set the analog value box back to its original place in the tab pane.



## PWM VALUE BOX

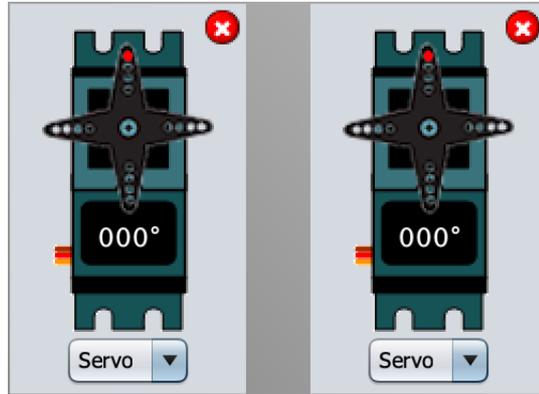
The PWM value box shows the value of each PWM pin of the Arduino. The cross button is used to set the PWM value box back to its original place in the tab pane.



## SERVO

There are 2 servos that can simulate a servo wheel with just one digital pin (D2-13) of the Arduino. The cross button is used to set the servo back to its original place in the tab pane.

The number of degrees (°) is visible in the servo. Click on the servo to make the servo smaller through removing the background and combobox. Use `servo.write()` and `servo.attach()`. Add the servo simulator library to use it.



## MOTOR WHEELS

There are 2 motor wheels that can turn forward and backwards with just 2 digital pins of the Arduino. To adjust the speed use one of the 6 PWM pins of the Arduino. The cross button is used to set the motor wheels back to its original place in the tab pane. By clicking on the wheel itself, you can hide the background and comboboxes.

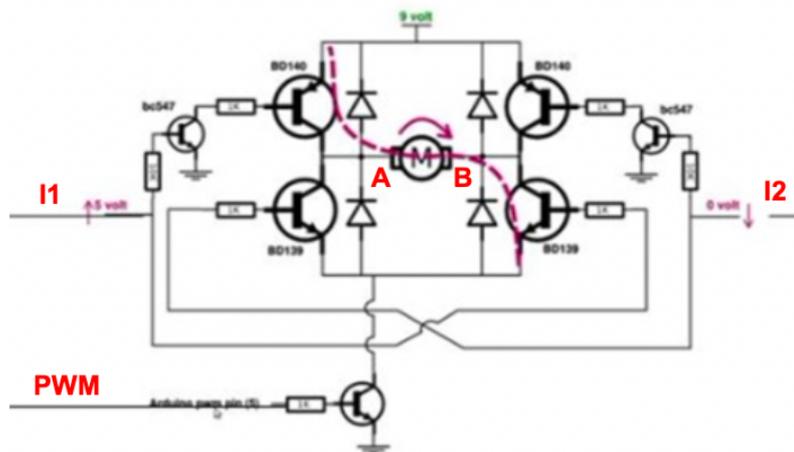
By turning the left wheel forward, you will turn your robot (example situation project) to the right because the right wheel doesn't turn. To drive forward, Run the 2 motors forward to drive forward. To control the motor wheels, you need to use `digitalWrite()` and `analogWrite()`.

See the Wheels and H-bridge examples to see how it works.

The operation is based on a robot with 2 wheels, which must be able to drive forwards / backwards as well as left and right. We use an H-bridge for turning the DC-motor.

Transistor H-bridge operations:

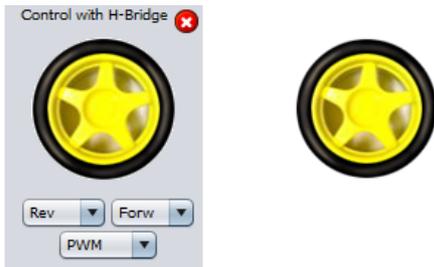
| <b>I1</b> | <b>I2</b> | <b>A</b> | <b>B</b> | <b>Motion</b> |
|-----------|-----------|----------|----------|---------------|
| Logic 0   | Logic 0   | 0V       | 0V       | Stop          |
| Logic 1   | Logic 0   | 9V       | 0V       | Forward       |
| Logic 0   | Logic 1   | 0V       | 9V       | Reverse       |
| Logic 1   | Logic 1   | 9V       | 9V       | Brake         |



**H-Bridge circuit**

## Arduino IO Simulator Drag & Draw

With the combo-box 'Rev' and 'Forw' (D2 to D13) can you simulate 'I1' and 'I2' (Reverse/Forward). With the combo-box 'PWM' (PWM 3 to PWM 11) can you simulate the speed.

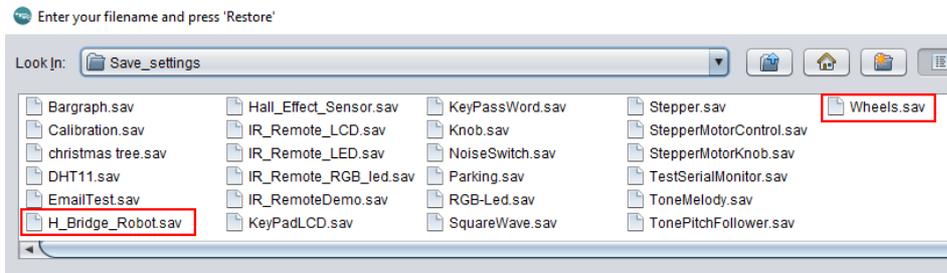


Click on the wheel removed the background

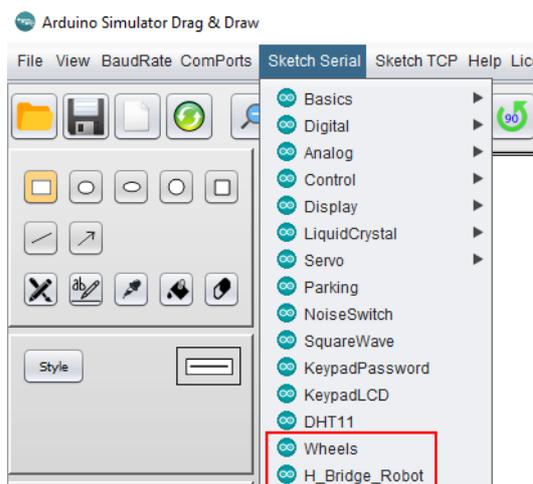
**Attention:** The wheels can only turn if a direction is selected (Rev or Forw)

Examples:

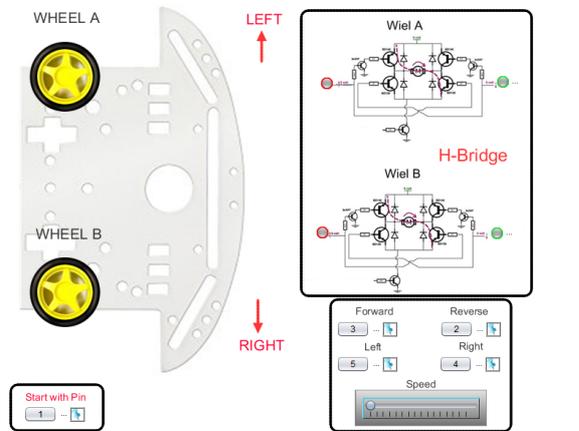
### Restore components and settings



### Load sketch into the Arduino board



## H\_Bridge\_Robot

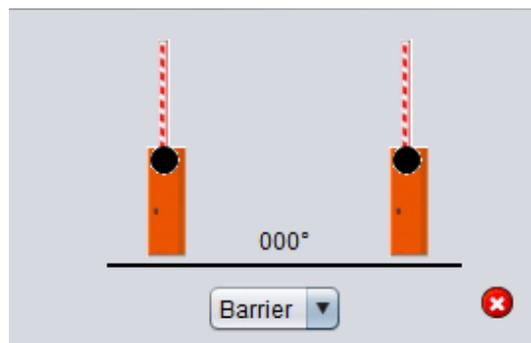


## Wheels



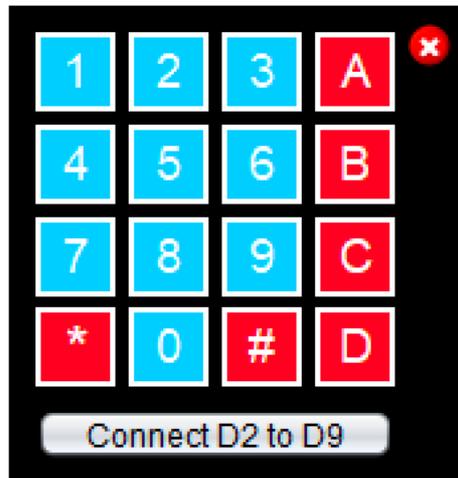
## BARRIER

The barrier works also like the servo with one signal (D2-13). The cross button is used to set the barrier back to its original place in the tab pane. By clicking on the barrier, itself you can hide the background and combobox.



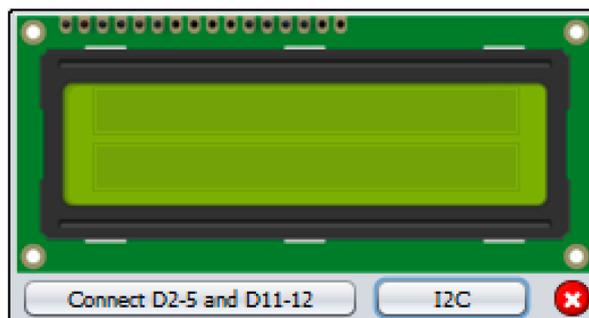
## KEYPAD

Connect the keypad to the Arduino by pushing the 'Connect D2 to D9' button. The cross button is used to set the keypad back to its original place in the tab pane.



## LCD DISPLAY

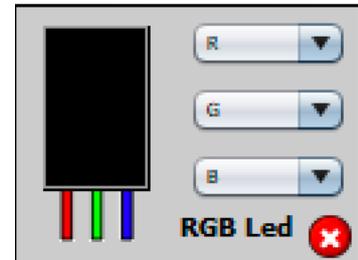
The LCD display can be connected to the Arduino by 2 modes. Connect D2-5, D11 and D12 or use the I<sup>2</sup>C option (A4-A5). The cross button is used to set the LCD display back to its original place in the tab pane. Add the simulator LiquidCrystalSim library to make it work with the simulator. Add the LiquidCrystalSim I2C library if you want to use the I2C connection.



## RGB LED

The RGB LED can be connected to the Arduino by 3 PWM pins. Connect 1 of the 6 PWM pins to each color value (R, G, B). The cross button is used to set the RGB LED back to its original place in the tab pane. Click on the LED itself to hide to settings. Each color represents a value of 0-255 which indicates a 0-255 color. Note: the 3 sliders are scaled to 0-1023.

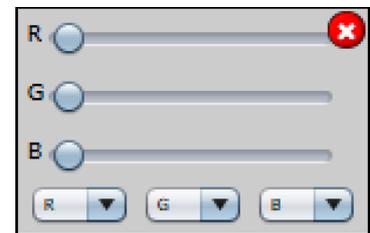
To control the RGB LED you use `analogWrite()`.



## RGB SLIDER

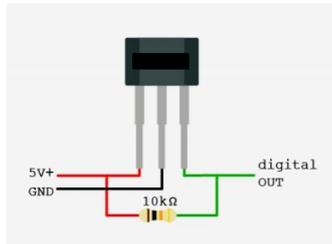
The RGB LED can be connected to the Arduino by 3 analog pins. Connect 1 of the 6 analog pins to each color value sliders (R, G, B). The cross button is used to set the RGB slider back to its original place in the tab pane.

Use the 'RGB\_Led' example and the 'RGB-Led.sav' (restore settings) to test the RGB LED with the sliders.



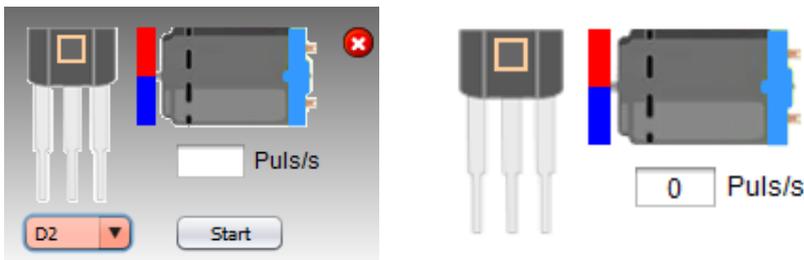
## HALL EFFECT SENSOR

Hall effect sensors have three pins: VCC(5V), GND, and Vout(Signal). The pinout of a Hall effect sensor is as shown below:



### Arduino Hall effect sensor pinout

The Vout or signal pin of the Hall effect sensor is connected to the Arduino's interrupt pin (digital pin 2 (D2))



Click on the sensor removed the background

### How Does It Work?

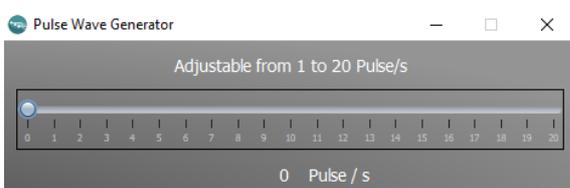
The Hall effect sensor works on the principle of the Hall effect, which states that whenever a magnetic field is applied in a direction perpendicular to the flow of electric current in a conductor, a potential difference is induced. This voltage can be used to detect whether the sensor is in the proximity of a magnet or not. The Arduino can detect this voltage change through its interrupt pin and determine whether the magnet is near the sensor or not. [1]

[https://en.wikipedia.org/wiki/Hall\\_effect\\_sensor](https://en.wikipedia.org/wiki/Hall_effect_sensor)

### Start simulation:

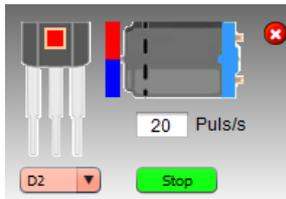
Connect the sensor to the Arduino by select pin 'D2'.

By pressing on the start button, we can rotate the magnet at a max speed of 20 pulses/second, this speed can be adjusted with a slider.



## Arduino IO Simulator Drag & Draw

In a text field you can see how many pulses/s the motor is running. You can stop the simulation by pressing the 'Stop' button.

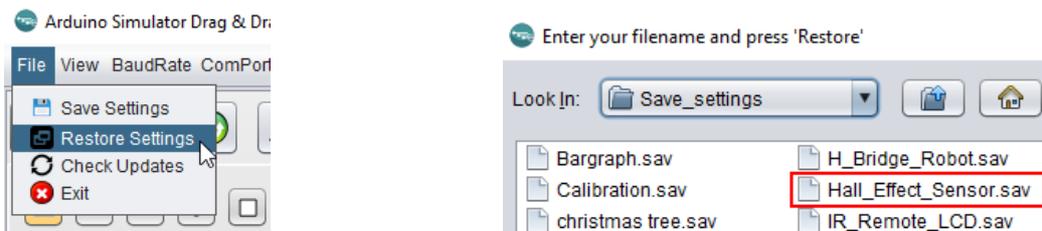


### Attention:

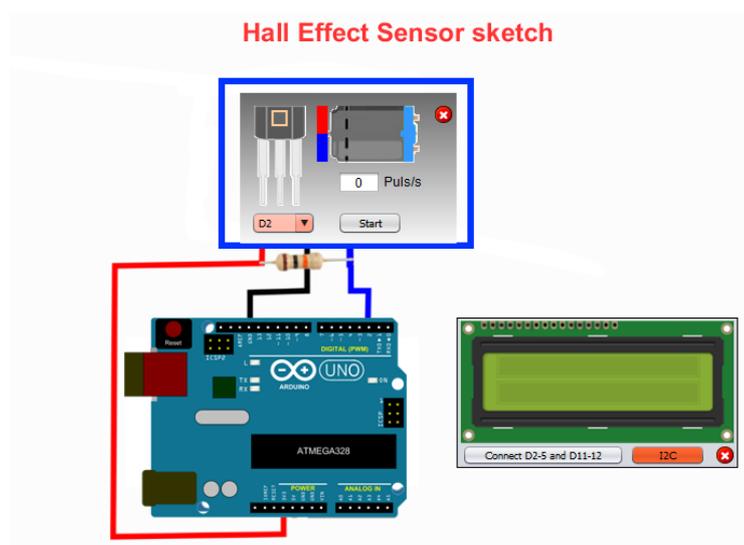
In the simulator there is an example to test the hall effect sensor with a 16x2 LCD display that print the values out.

Start simulation with this example:

Load the necessary components by click on 'Restore Settings' and load the file 'Hall\_Effect\_Sensor.sav' into you're Arduino board.



The necessary components are loaded on the screen and are already connected to the simulator.



## Arduino IO Simulator Drag & Draw

The components are projected on a drawing with an Arduino, the Arduino drawing with lines and resistor is only intended for the visual effect.

Load the sketch 'HallEffectSensor' into the Arduino (Sketch Serial).

For running this sketch you need 2 libraries:

- **SimulatorProgram.h**
- **LiquidCrystal\_I2C\_Sim.h**

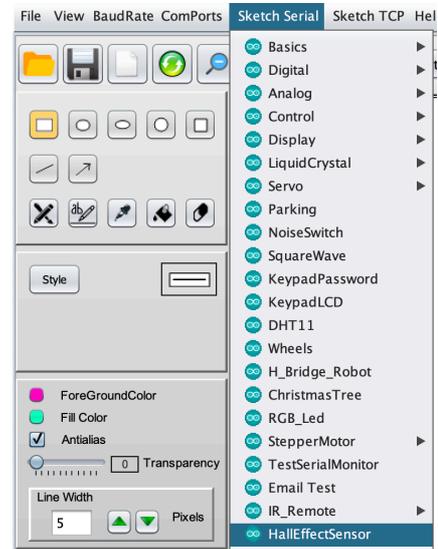
```
Hall_Effect_Sensor | Arduino 1.8.10
Bestand Bewerken Schets Hulpmiddelen Help

Hall_Effect_Sensor
//*****
//*****
//***** Cc
//*****

#include <SimulatorProgram.h>
#include <Wire.h>
#include <LiquidCrystal_I2C_Sim.h>

// digital pin 2 is the hall pin
int hall_pin = 2;
// set number of hall trips for RPM reading (higher improves accuracy)
float hall_thresh = 10.0;

LiquidCrystal_I2C lcd(0x21, 16, 2);
```



The library 'SimulatorProgram.h' is already present (is automatically installed by the program). The library 'LiquidCrystal\_I2C\_Sim.h' if not already installed, you still must install it yourself.

To install a library, see section 'Code changes'.

For proper operation it is necessary to set the BaudRate as high as possible, so set it to "115200".

```
void setup() { |
    Serial.begin(115200);
    inString.reserve(10);
```

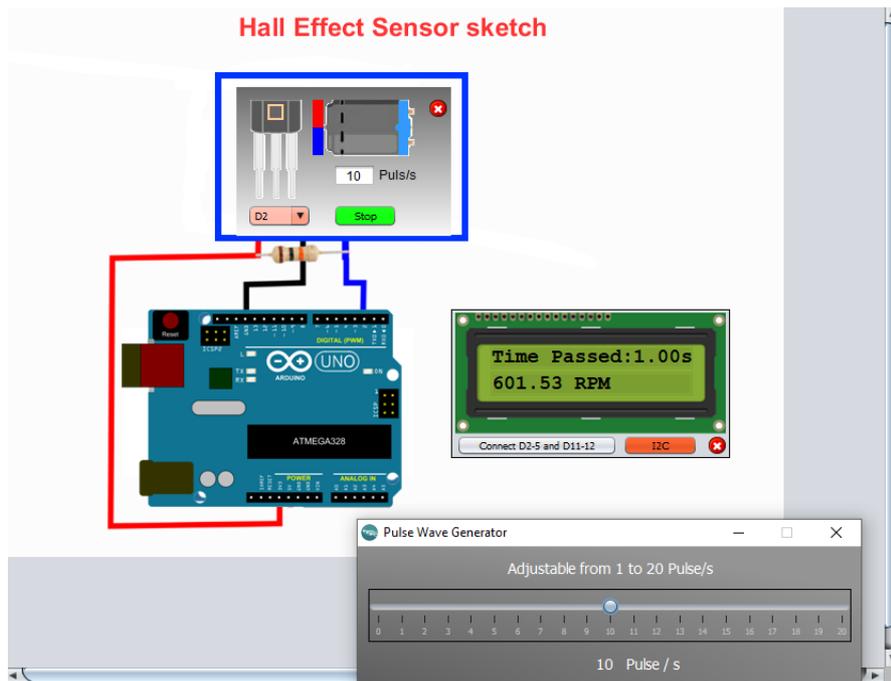
With the 'hall\_tresh' variable you set the desired number of pulses per revolution.

This means that for 10 pulses /s the motor speed is 600 rpm.

The sketch calculates this for you and shows this on the LCD.

```
// digital pin 2 is the hall pin
int hall_pin = 2;
// set number of hall trips for RI
float hall_thresh = 10.0;
```

Example:



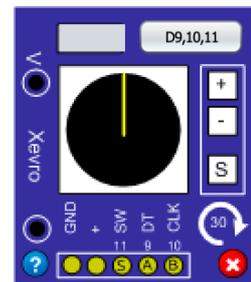
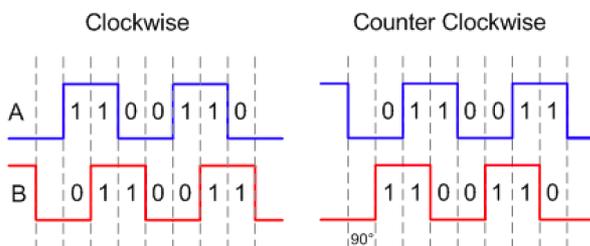
**Attention:** Do not forget to set the BaudRate to "9600" at the end of the simulation to make sure you don't forget it for other simulations.

## ROTARY ENCODER

The rotary encoder module can simulate a 30 steps rotating potentiometer with a button. The rotary encoder can give 30 pulses within 1 full rotation. The button + and – buttons will increase and decrease the position. The 'S' will push the button and sets an action like you programmed, for example confirming an action. If you keep pressing the +/- buttons, the encoder turns at 1 pulse/s (second) forward.

The rotary encoder can be connected to 3 pins, we can use digital pin 9, 10 and 11. When you click the button it can reset the counter and sets the encoder in its start position. There are 3 examples available to test in the sketches list as well as 3 restore files with the settings in. The simulated rotary encoder can't work with interrupted processes due to serial communication usage together with the simulator software.

### A visualization of the data pin statuses:



Output A (DT) = pin 9 on the Arduino (I09)

Output B (CLK) = pin 10 on Arduino (I10)

Serial code = ROTAB (open the serial monitor to track the codes)

More information can be found on this website:

<https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/>

## IR REMOTE CONTROL

With the IR remote control simulator part, it is possible to simulate an IR LED and remote controller by using the IRremote library that we adjusted so it works together with the simulator. The Arduino instructions for the IR remote are the same as for the original library, you only have to change the library in order to use the simulated IR remote controls.

Go to the tab page 'IR-Remote' by pressing the "More IO" button to show more IO parts:

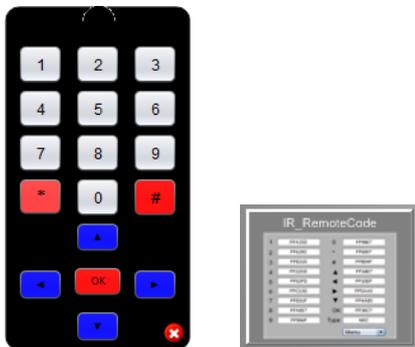


Here you can use various IR tools:

### 1. Remote controller

With this remote controller you can send a code to the Arduino by pressing the keys on the controller, you can see the IR led flashing. (standard NEC code)

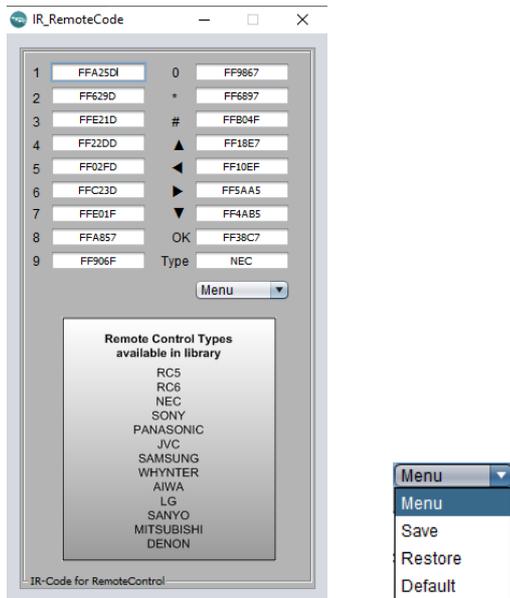
It is possible to link your own code to the keys.



By pressing the button 'IR\_RemoteCode' a window will appear with the default codes (standard NEC), but you can adjust these codes according to your needs. (Image on next page).

With the combobox 'Menu' you can Save and Restore your code to a file, or you can choose the default code (NEC).

In the box under the 'Menu' you will see the names of the remote control types that are in the library.



## 2. IR receiver

With the combobox you can connect the receiver to the desired Arduino pin, for the IR receiver is this normally D7 or D11.



By pressing a key on the simulator remote controller or a real remote controller, you can see the receiver flashing.

## 3. IR transmitter

With the combobox you can connect the transmitter to the desired Arduino pin, for the transmitter is this normal D3.



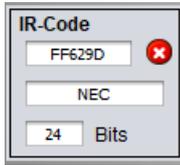
If the Arduino sends a signal on D3 or by pressing a key on a real remote controller, the IR-LED will start flashing.

If you give a mouse click on the LED then you see only the LED.



#### 4. IR-codebox

In this box you see the received code in Hex format, number of bits and type of remote controller.



#### 5. Blinkled13

In the void setup you can enable/disable the blinking of pin D13 on IR processing with the instruction ``irrecv.blink13(true/false)``.



If you give a mouse click on the '...' then you see only the LED.

**Note:** With 'View'- 'Collapse IO' you can adjust all the objects in one click.

## USE THE IR REMOTE LIBRARY

### 6. Install the library 'IRremoteSim.zip'

We have adapted the 'IRremote' library and rename him to 'IRremoteSim' so that we can work with the same instructions.

1 instruction has been added to the library: `IRrecv::SimRcvIRcode` (long value, int bit, int type)

In the directory `C:\Program Files (x86)\ArduinoSimulator\libraries\IRremoteSim` you can find the '**IRremoteSim.zip**' file, you need to install the IR library to use the IR tools, this can be done by installing the library in the simulator.

How can you install the library '**IRremoteSim.zip**':

Click on the 'libraries' tab in the tools bar and select the IRremoteSim folder. Select the .zip file and install the library. This will add the library to the Arduino/libraries folder and can now be found in the Arduino IDE libraries list.

### 7. Programming Arduino IR-code

```
#include "SimulatorProgram.h"
#include <IRremoteSim.h>

int RECV_PIN = 7;
IRrecv irrecv(RECV_PIN);
decode_results results;
IRsend irsend;

void setup()
{
  Serial.begin(9600);
  inString.reserve(10);

  // In case the interrupt driver crashes on setup, give a clue
  // to the user what's going on.
  Serial.println("Enabling IRin");
  irrecv.enableIRin(); // Start the receiver
  Serial.println("Enabled IRin");
  irrecv.blink13(true); //Enable blinking the LED when during reception
}
```

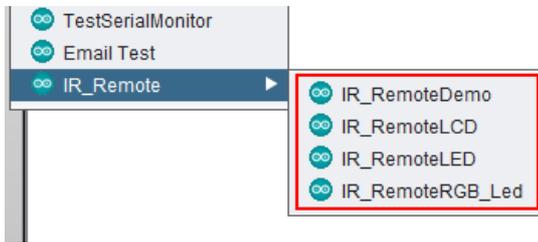
#### Attention:

If you write a program for the Arduino, the following program code must always be at the top in the void loop(), this program code sends the data from the simulator to the library 'IRremoteSim'.

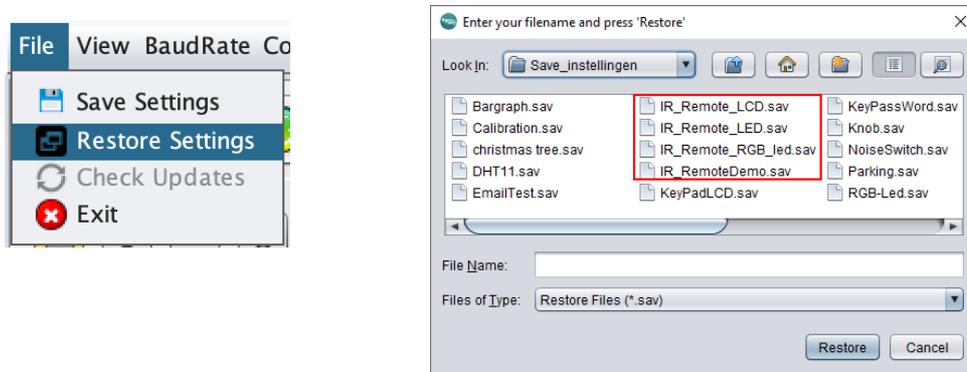
```
void loop() {
  // ***** Simulator IRremote code , NOT REMOVE: *****
  if (Sim_IRcode != 0){
    irrecv.SimRcvIRcode(Sim_IRcode, Sim_IRbits, Sim_IRtype);
    Sim_IRcode = 0;
  }
}
```

### 8. There are 4 program examples available:

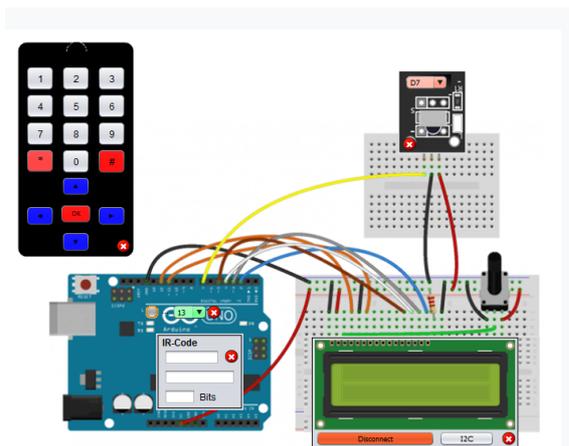
Arduino IR sketches (choose Sketch Serial):



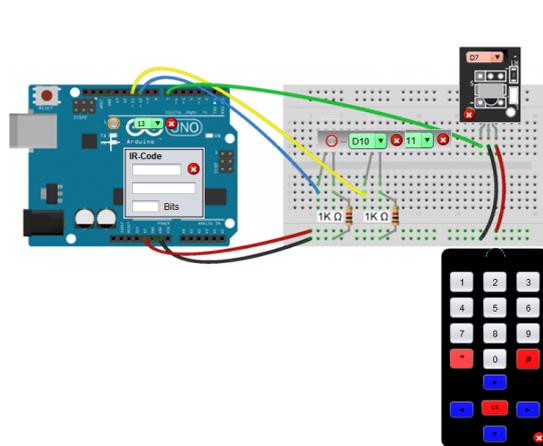
Restore the pre-programmed examples:



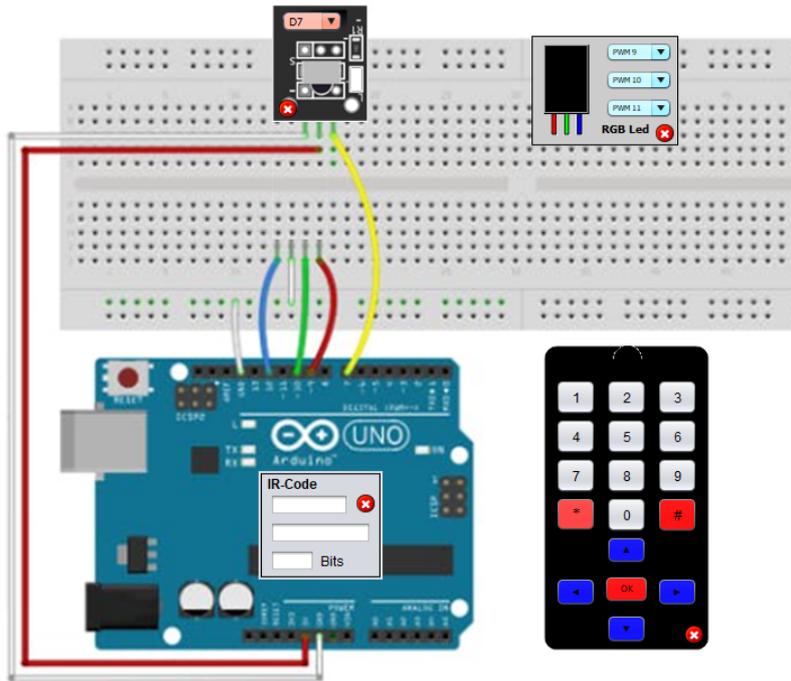
### IR\_Remote\_LCD



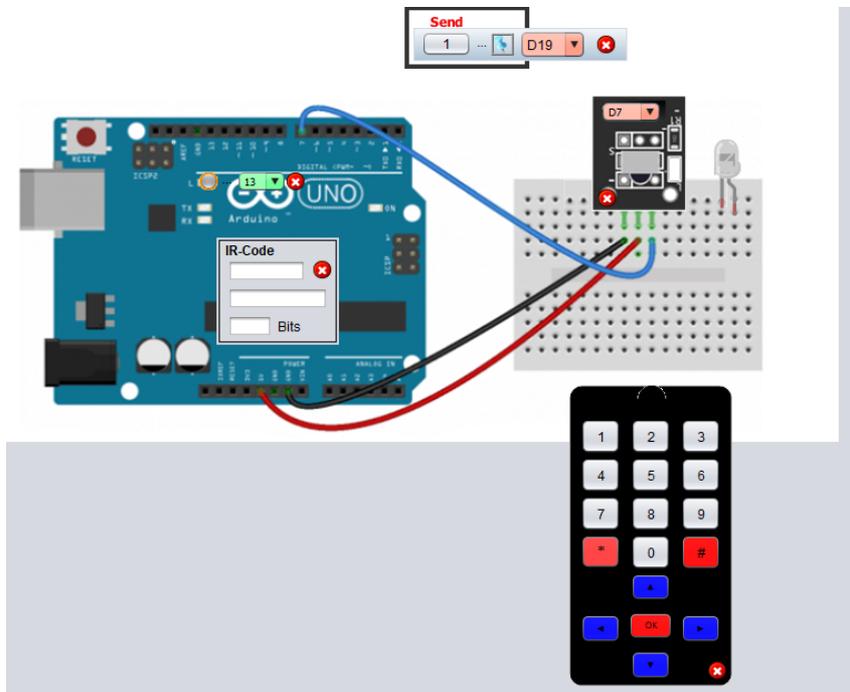
### IR\_Remote\_LED



### IR\_Remote\_RGB\_Led



### IR\_RemoteDemo



## STEPPER MOTOR 28-BYJ48

The Stepper motor can be connected to the Arduino by 2 or 4 digital pins (8, 10, 9 and 11). The driver module (ULN2003) is processed in the simulator. There are 4 LEDs that indicate which coil is energized on which specific moment in time. The amount of steps and degrees is shown next to the 4 LEDs. Every simulation requires the change of a command in the Arduino IDE, with the stepper motor can you still use the original code. You only need to replace the library and this will simulate. Read more about this under 'Stepper Library'.

The cross button is used to set the stepper motor back to its original place in the tab pane. You can find the stepper motor under the 'More I/O' button.

**The simulator stepper motor has 2 input fields:**

- **Sim\_SPEED**
- **Sim\_STEPS**

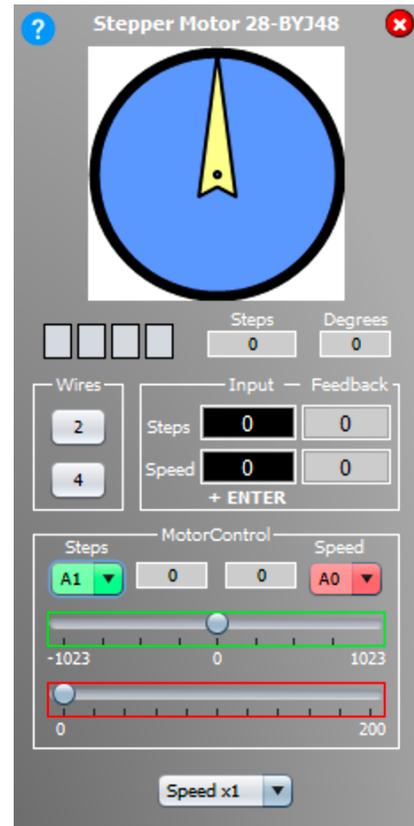
Steps = 1 to 2048 (360°)  
Speed = 1 to 200

Input Steps -> Sim\_STEPS  
Input Speed -> Sim\_SPEED

```
stepper.step(Sim_STEPS);  
  
stepper.setSpeed(Sim_SPEED);
```

Use the 'Stepper.sav', 'StepperMotorControl.sav' and the 'StepperMotorKnob.sav' example (restore settings) to test the stepper motor.

If you are using the stepper motor and also the serial monitor together, it's possible you will get a warning that will tell you to close the serial monitor and reset the Arduino because you are losing steps.

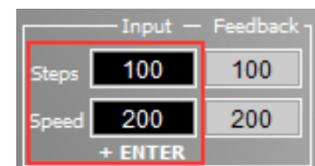


## INPUT/FEEDBACK

If you want to specify the number of steps or speed you must press the 'enter' key after typing in. The Arduino will send the steps and speed back to the simulator to show you it is received (Feedback).

**Note:** Make sure you click on the 'enter' key after adding the value for Sim\_SPEED and Sim\_STEPS

### Speed



It's possible to speed up the stepper motor 2 times faster because of the serial port speed, the real stepper motor is a little bit faster. With even numbers the steps will be exactly the same. With uneven numbers there will be a loss of 2 steps.

## STEPPER LIBRARY

To control the stepper motor you use can use the same commands as before but just change the library to our provided library 'StepperSim.zip' and add them to the Arduino IDE.

The original library has been changed so that it works together with the simulator 'Stepper\_Sim.zip'.

You can find it on this directory:

**C:\Program Files (x86)\ArduinoSimulator\Library** or

**C:\Program Files\ArduinoSimulator\Library**

With the Arduino IDE, you use 'Include library' and select 'Add .Zip' Library to install the library 'Stepper\_Sim.zip'.

You can also always download the library here:

<https://xevro.be/onewebmedia/Arduino%20Simulator%20Drag%20&%20Drop%20Userguide/Stepper%20motor%20guide%20images/StepperSim.zip>

In your Arduino sketch write '#include <Stepper\_Sim.h>' to add the stepper library to your project.

**The stepper motor commands are the same as the ones in the stepper library provided by Arduino.**

stepper.**setSpeed**(value between 0-200);

Stepper(STEPS, pin1, pin2);                      For 2 coil winding

Stepper(STEPS, pin1, pin2, pin3, pin4);        For 4 coil winding

setSpeed(rpm);

step(STEPS);

**More information about the stepper commands:**

<https://www.arduino.cc/en/Reference/Stepper>

## CALCULATE THE STEPS PER REVOLUTION FOR THE STEPPER MOTOR

It's important to know how to calculate the steps per revolution for the stepper motor because only then you can program it effectively.

In the simulator, we will be operating the motor in 4-step sequence so the stride angle will be  $11.25^\circ$  since it is  $5.625^\circ$  for 8 step sequence it will be  $11.25^\circ$  ( $5.625 * 2 = 11.25$ ). You can also choose for a 2-step sequence, but you will need to change the coil pins in the sketch.

**Stepper stepper (STEPS, 8, 10);**

Steps per revolution =  $360/\text{step angle}$   
 $360/11.25 = 32$  steps per revolution

The number of steps per revolution for our stepper motor was calculated at 32; therefore we enter this as shown in the line below:

**#define STEPS 32**

Next, you have to create instances in which we specify the pins to which we have connected the stepper motor.

**4 coil wires:** Stepper stepper (STEPS, 8, 10, 9, 11);

**2 coil wires:** Stepper stepper (STEPS, 8, 10);

**Note:** The pins number are disordered as 8, 10, 9, 11 on purpose. You have to follow the same pattern even if you change the pins to which your motor is connected.

Since we are using the Arduino stepper library, we can set the speed of the motor using the below line. The speed can range between 0 to 200 for the 28-BYJ48 stepper motor.

**stepper.setSpeed(200);**

To make the motor move one step we can use the following line.

**stepper.step(val);**

The number of steps to be moved will be provided by the variable 'val'. Since we have 32 steps and 64 as the gear ratio we need to move 2048 ( $32*64 = 2048$ ), to make one complete rotation.

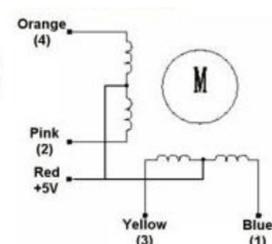
The value of the variable 'val' can be entered by the user using the serial monitor.

**Coil wires: 2 or 4**

Select in the simulator how many windings (2 or 4) your stepper motor has.

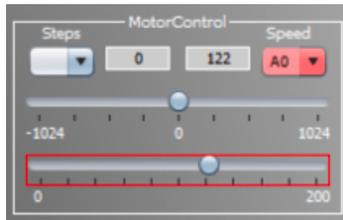
For 4 coil winding: `Stepper stepper(STEPS, 8, 9, 10, 11);`

For 2 coil winding: `Stepper stepper(STEPS, 8, 9);`



## MOTORCONTROL

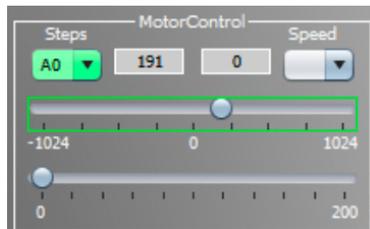
To adjust the speed of the motor you can use the provided slider and select a pin from A0-A5.



### Arduino code:

```
int motorSpeed = map(sensorReading, 0, 1024, 0, 100);
int sensorReading = AnalogRead(A0);
myStepper.setSpeed(motorSpeed);
```

To adjust the steps of the motor you can use the provided slider and select a pin from A0-A5.



### Arduino code:

```
int val = AnalogRead(0);
stepper.step(val - previous);
previous = val;
```

**Note:** If you can't start the stepper motor, you can click on the reset button in the simulator and it will send all the start values again. Disconnect the coil (2 or 4) and then reconnect it again.

## MAX7219 LED DOT MATRIX DISPLAY

In this tutorial, you will learn how to control a MAX7219 LED dot matrix display with Arduino. The code in this tutorial can be used for 8×8, 8×32, and even larger displays.

For this tutorial, I will be using the MD\_Parola in combination with the MD\_MAX72XX Arduino library. These libraries make displaying scrolling text and other animations super easy.

### ABOUT THE MAX7219 LED DRIVER

The MAX7219 LED driver can be used to control 7-segment displays up to 8 digits, bar-graph displays, or 64 individual LEDs. The driver communicates with the Arduino through SPI so you only need three wires to control the display.

Since the MAX7219 can control a maximum of 64 LEDs, the maximum size dot matrix display it can drive is 8×8 pixels. However, you can daisy chain multiple drivers and matrices together and easily control much larger displays like 8×32, 8×64, or even bigger. Still, you only need three wires to control all of the ICs so you need very few I/O pins of the Arduino. [2]

### HOW TO CONNECT THE DOT MATRIX DISPLAY TO THE ARDUINO

The MAX7219 LED display driver communicates with the Arduino through SPI (Serial Peripheral Interface). To learn more about this data protocol, please see: <https://www.arduino.cc/en/reference/SPI> on the Arduino website.

With an SPI interface there is always one master device (the Arduino) that controls the peripheral devices (also known as slaves). You can control the display either through the Arduino's AVR microcontroller hardware SPI interface or three arbitrary digital pins (software SPI).

The hardware SPI pins (MOSI, MISO, and SCK) are at a specific location on each Arduino board. This interface is faster than using software SPI, but you will need to use the following fixed output pins:

## HARDWARE SPI PIN LOCATIONS ARDUINO UNO

CLK\_PIN 13 (SCK)

DATA\_PIN 12 (MISO)

CS\_PIN 11 (MOSI)

Note that the MOSI, MISO, and SCK pins are also at a consistent physical location on the 6-pin ICSP header:



To control MAX7219 displays you only need to make three connections:

**MOSI** (Master Out Slave In) connected to **DIN** – The Master line sending data to the peripherals.

**SCK** (Serial Clock) connected to **CLK** – The clock pulses which synchronize data transmission generated by the master.

**SS** (Slave Select) connected to **CS** – The pin on each device that the master can use to enable and disable specific devices.

### Note:

We use the software 'SPI.h' library with the simulator, you can connect DIN, CS, and CLK to any of the digital pins on the Arduino. You just need to specify the pin numbers in the setup of the Arduino code (see example below).

To control the MAX7219 display we will be using two awesome Arduino libraries created by Marco Colli from MajicDesigns. The **MD\_Parola** library can be used to create many different text animations like scrolling and sprite text effects. This library depends on the **MD\_MAX72XX** library which implements the hardware functions of the LED matrix.

This are some functions and features of the library:

- Left, right, or center text justification
- Text scrolling with entry and exit effects
- Control display parameters and animation speed
- Multiple virtual displays (zones) in each string of LED modules
- Support for hardware SPI interface
- User-defined fonts and/or individual characters substitutions
- Support for double-height displays
- Support for mixing text and graphics on the same display

## Arduino IO Simulator Drag & Drop

To make the libraries work with the Arduino simulator, we had to make a small adjustment and rename the library. Nothing has changed in the way of operation, just an adjustment to send the data serially to the simulator.

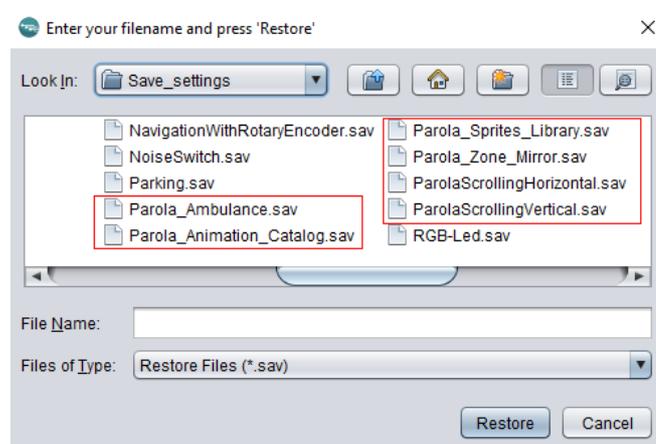
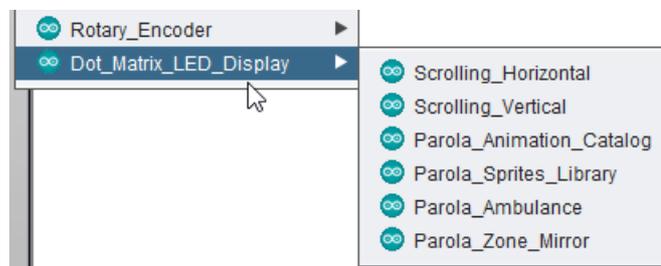
The new libraries are called **MD\_Parola\_Sim** and **MD\_MAX72XX\_Sim**.  
The simulator has 2 FC16 modules.

The FC-16 8x8 or 8x16 module:



- The radio button 'On' makes the matrix available
- The radio button '90°' rotates the matrix vertically
- By pressing the 'On' button Off, you can clear the matrix
- By pressing the red cross button, the matrix goes back to its starting position
- For correct operation, matrix2 must become before matrix1

By clicking on the border you can drag and drop the matrix in the simulator field.  
There are 6 sketches available with have a saved profile that can be restored.



# Arduino IO Simulator Drag & Draw

These 6 examples are available to try:

Parola\_Ambulance



Parola\_Animation\_Catalog



Parola\_Animation\_Catalog



Parola\_Zone\_Mirror



Parola\_Scrolling\_horizontal



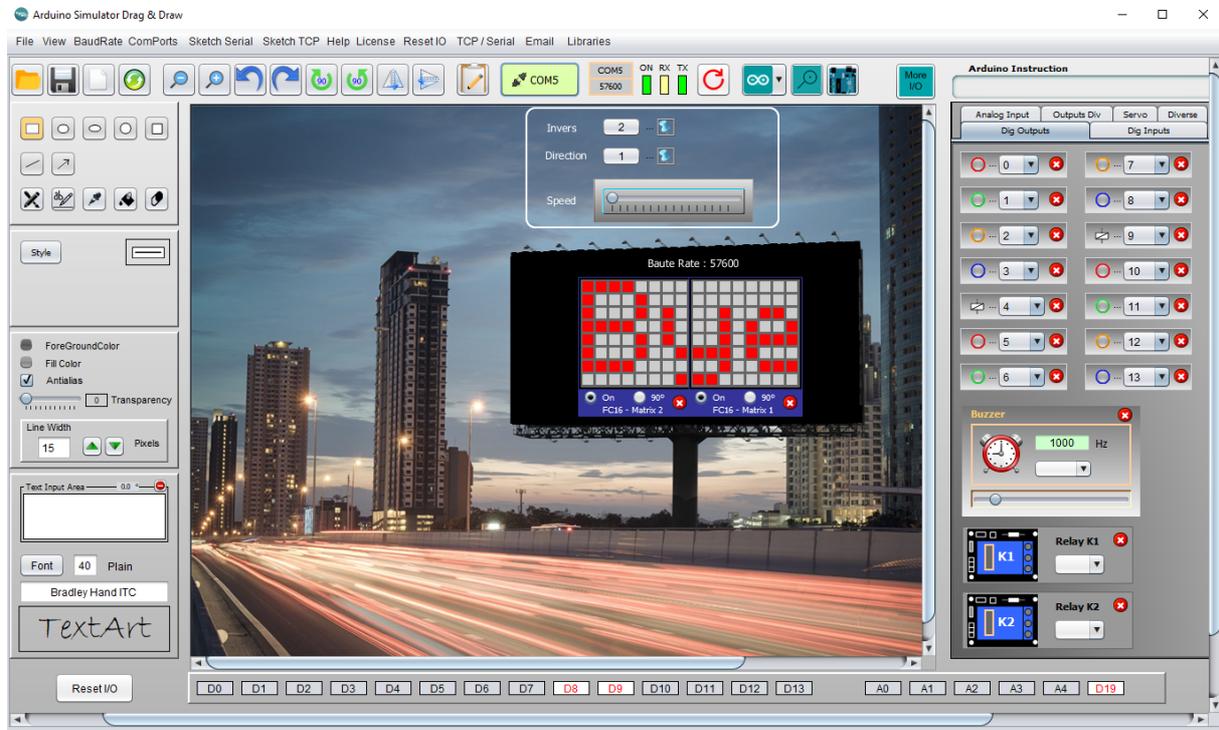
Parola\_Scrolling\_Vertical



## Arduino IO Simulator Drag & Draw

You can find all the Parola sketches on the C:\Program Files (x86)\ArduinoSimulator\ArduinoSketches

### EXAMPLE SKETCH "PAROLA\_SCROLLING\_HORIZONTAL"



## HARDWARE CONFIGURATION IN ARDUINO

When setting up the display in your Arduino code you need to set the **HARDWARE\_TYPE** to **FC16\_HW** and specify the number of devices you have connected, in this case 2.

An 8×8 matrix counts as 1 device, so if you want to control an 8×16 module you need to set **MAX\_DEVICES** to 2 (an 8×16 display contains 2 MAX7219 ICs).

```
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 2
```

## HOW THE CODE WORKS

The first step is to include all the required Arduino libraries. As mentioned before, the **MD\_MAX72XX\_Sim** library implements the hardware functions of the LED matrix and the **MD\_Parola\_Sim** library text effects. You will also need to include the SPI library and the Simulator library, which comes pre-installed in the Arduino IDE. The **SPI** library is used for the Serial Peripheral Interface communication between the display and the Arduino.

// Include the required Arduino libraries:

```
#include <MD_Parola_Sim.h>
#include <MD_MAX72xx_Sim.h>
#include <SPI.h>
#include <SimulatorProgram.h>
```

Next, we need to specify which hardware we are using. I used 2 standard 8×8 display (also known as **FC-16**), I set the **HARDWARE\_TYPE** to **FC16\_HW**. The number of MAX7219 ICs in an 8×16 display is 2 so I set **MAX\_DEVICES** to **2**.

Lastly, I defined to which pin the CS pin of the display is connected (output pin 10 in this case).

Because we work with software SPI, we also need to define the data and clock output pins and pass these as parameters when setting up the sketch.

```
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 2
#define CLK_PIN 13
#define DATA_PIN 11
#define CS_PIN 10
```

The statement `#define` is used to give a name to a constant value. The compiler will replace any references to this constant with the defined value when the program is compiled. So everywhere you mention `CS_PIN`, the compiler will replace it with the value 10 when the program is compiled.

After this, a new instance of the `MD_Parola` class is created with the function `MD_Parola()`.

This function needs five parameters: **Hardware type, Data\_Pin, CLK\_Pin, CS\_Pin, Max\_Devices.**

Note that I have called the MD\_Parola object '**P**' but you can use other names as well. You will need to change '**P**' to the new name in the rest of the sketch.

```
// SOFTWARE SPI  
MD_Parola P = MD_Parola(HARDWARE_TYPE, DATA_PIN, CLK_PIN, CS_PIN, MAX_DEVICES);
```

In the setup section of the code, the first 3 lines are need for the simulator:

Initialize the serial connection (BaudRate 57600)

Stringbuffer simulator (25)

RealIO\_Connect() for working with the real in- or outputs on the board

```
void setup()
{
  Serial.begin(57600);
  inString.reserve(25);
  RealIO_Connect(); // Used for connection between arduino and real IO
}
```

The next lines are using for the control functions (button and slider)

```
// set to 1 if we are implementing the user interface pot, switch, etc
#define USE_UI_CONTROL 1

#if USE_UI_CONTROL
  pinMode(SPEED_IN, INPUT);
  pinMode(DIRECTION_SET, INPUT);
  pinMode(INVERT_SET, INPUT);
  doUI();
#endif // USE_UI_CONTROL
```

In the setup section of the code, we first initialize the object with the function begin()

```
P.begin();
```

At the end of the setup section, we specify how we want to display the text with the function displayText(curMessage, scrollAlign, scrollSpeed, scrollPause, scrollEffect, scrollEffect). This function takes 5 arguments.

The first parameter is the text string, in this case curMessage.

The second argument sets the alignment of the text during the optional pause. In this case PA\_LEFT.(other are PA\_RIGHT , PA\_CENTER)

The third and fourth arguments set the speed of the animation and pause time respectively. The speed of the display is the time in milliseconds between animation frames. The lower this time the faster the animation. If you want to pause the text in between the in and out animation, you can set the pause time in milliseconds. If the pause is zero so the text scrolls continuously.

Next, the in and out effects are specified. In this case I used PA\_SCROLL\_LEFT for both. See the example below for other text effects.

```
uint8_t scrollSpeed = 10; // default frame delay value
textEffect_t scrollEffect = PA_SCROLL_LEFT;
textPosition_t scrollAlign = PA_LEFT;
uint16_t scrollPause = 20; // in milliseconds
```

## Arduino IO Simulator Drag & Draw

```
P.displayText(curMessage, scrollAlign, scrollSpeed, scrollPause, scrollEffect, scrollEffect);
```

With the function 'doUI' you can use buttons, sliders for manipulating the matrix.

```
// set to 1 if we are implementing the user interface pot, switch, etc
#define USE_UI_CONTROL 1
```

With a slider from the simulator you can change the speed, but the effect is not great due to the slow serial communication.

```
#if USE_UI_CONTROL

void doUI(void) {
  // set the speed if it has changed
  {
    int16_t speed = map(analogRead(SPEED_IN), 0, 1023, 10, 150);
    if ((speed >= ((int16_t)P.getSpeed() + SPEED_DEADBAND)) ||
        (speed <= ((int16_t)P.getSpeed() - SPEED_DEADBAND)))
    {
      P.setSpeed(speed);
      scrollSpeed = speed;
      PRINT("\nChanged speed to ", P.getSpeed());
    }
  }
}
```

With a button from the simulator you can change the direction or invert (it is not possible to invert a sprite with the simulator).

```
if (digitalRead(DIRECTION_SET) == HIGH) // SCROLL DIRECTION
{
  PRINTS("\nChanging scroll direction");
  scrollEffect = (scrollEffect == PA_SCROLL_LEFT ? PA_SCROLL_RIGHT : PA_SCROLL_LEFT);
  P.setTextEffect(scrollEffect, scrollEffect);
  P.displayClear();
  P.displayReset();
}

if (digitalRead(INVERT_SET) == HIGH) // INVERT MODE
{
  PRINTS("\nChanging invert mode");
  P.setInvert(!P.getInvert());
}
```

In the loop section, you only need two functions to create a scrolling text display.

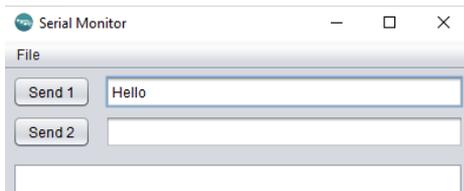
First, we use the function `displayAnimate()` in an if statement. This function animates the display using the currently specified text and animation parameters and returns true when the animation has finished.

When the animation has finished, we reset the display with the function `displayReset()` so the text is displayed in a loop.

## Arduino IO Simulator Drag & Draw

```
if (P.displayAnimate())
{
  if (newMessageAvailable)
  {
    strcpy(curMessage, newMessage);
    newMessageAvailable = false;
  }
  P.displayReset();
}
```

With the function 'readSerial' you can change the text online with help of the Serial monitor.



All the data from the simulator can be seen on the matrix if you activate the 'readSerial' function in the void loop(). (button ex. I091111)

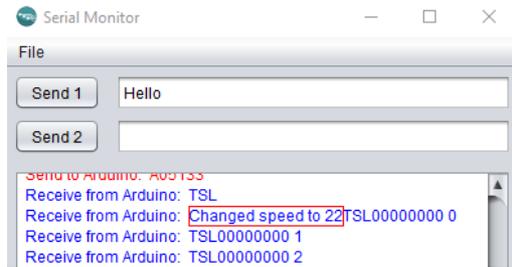
```
void readSerial(void)
{
  static char *cp = newMessage;
  //static String *cp = newMessage;
  while (Serial.available())
  {
    /*cp = (char)Serial.read();
    if ((*cp == '\n') || (cp - newMessage >= BUF_SIZE-2)) // end of message character or full buffer
    {
      *cp = '\0'; // end the string
      // restart the index for next filling spree and flag we have a message waiting
      cp = newMessage;
      newMessageAvailable = true;
    }
    else // move char pointer to next position
      cp++;
  }
}
```

With the debugger you can make your changes visible in the Serial Monitor

```
// Turn on debug statements to the serial output
#define DEBUG 1

#if DEBUG
#define PRINT(s, x) { Serial.print(F(s)); Serial.print(x); }
#define PRINTS(x) Serial.print(F(x))
#define PRINTX(x) Serial.println(x, HEX)
#else
#define PRINT(s, x)
#define PRINTS(x)
#define PRINTX(x)
#endif
```

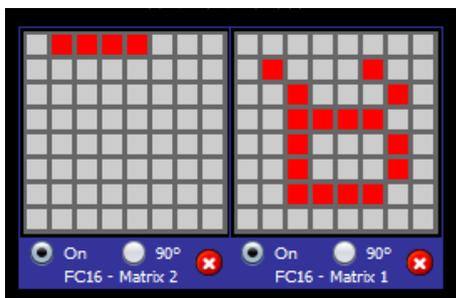
Example change speed with slider:



Attention: It is possible that the 'PRINTS' from the debugger the text disturbs.

A print instruction in the setup disturbs also the start of the text , so do not place printing instructions in the setup.

It looks like this:



## OTHER TEXT EFFECTS

The library **MD\_Parola\_Sim** includes several other text effects that you can use:

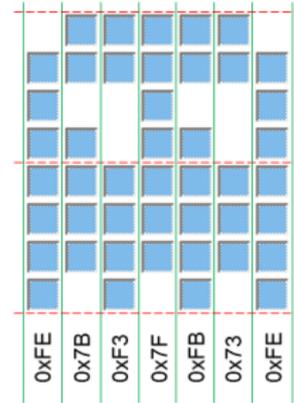
- PA\_PRINT,
- PA\_SCROLL\_LEFT,
- PA\_WIPE,
- PA\_SCROLL\_UP\_LEFT,
- PA\_SCROLL\_UP,
- PA\_SCROLL\_UP\_RIGHT,
- PA\_BLINDS,
- PA\_SCROLL\_DOWN\_LEFT,
- PA\_WIPE\_CURSOR,
- PA DISSOLVE,
- PA\_SCROLL\_DOWN\_RIGHT,
- PA\_SCROLL\_RIGHT,
- PA\_SLICE,
- PA\_SCROLL\_DOWN

More information on this link about the Parola library:

[https://majicdesigns.github.io/MD\\_Parola/page\\_software.html](https://majicdesigns.github.io/MD_Parola/page_software.html)

## MAKE YOUR OWN SPRITES

Each frame is defined by a sequence of numbers that encode the columns of the bitmap. In the example on the left (a Pacman 'ghost' character), each column of bits is represented by the hexadecimal number at the base of the bitmap. The least significant bit is at the top of the bitmap. If the sprite has a front and rear, the bitmap should be defined for the sprite moving to the right. The library will mirror reverse the image when it moves left.



The sprites are essentially defined in the same way as the character font and the same tools can be used to define the data for the sprite bitmap. Note that, like the font definition, the sprite is stored in PROGMEM to save RAM space.

A sprite has at least one frame. If more than one frame is required, a similar definition is created for each frame of the animation, and a data table constructed defining the animated sprite, as shown in the code snippet below, which is for the ghost character with shifting eyes.

Two convenience constants are used to define the sprite, one for the width (number of bytes) data for one sprite and the other for the number of frames contained in the animation. The total number of bytes required is the width \* number of frames. [3]

More information on <https://arduinoplusplus.wordpress.com/2018/04/19/parola-a-to-z-sprite-text-effects/>

```
const uint8_t F_GHOST = 2;
const uint8_t W_GHOST = 7;
static const uint8_t PROGMEM ghost[F_GHOST * W_GHOST] =
{
    0xfe, 0x7b, 0xf3, 0x7f, 0xfb, 0x73, 0xfe,
    0xfe, 0x73, 0xfb, 0x7f, 0xf3, 0x7b, 0xfe,
};
```

To ensure smooth animations, you should remember that once the last frame is reached, it will loop back to the first. Avoid discontinuities between the two ends of the data table.

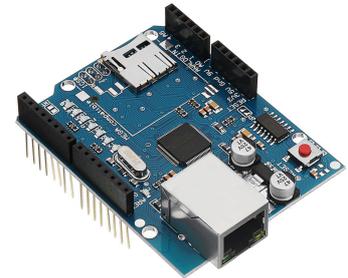
# ARDUINO TCP ETH-WIFI CONNECTION

The Arduino simulator can connect to an Arduino with an ethernet or WIFI wireless connection. We used an Arduino W5100 ethernet shield for the Arduino UNO and an Arduino UNO WIFI rev2.

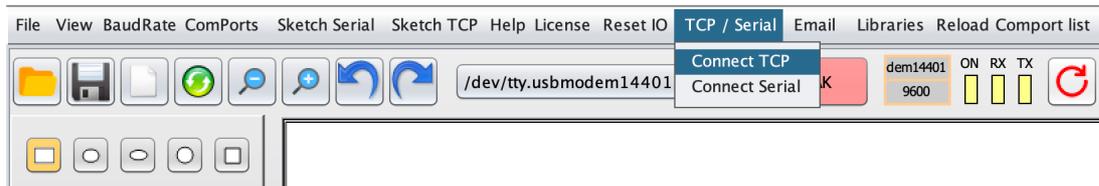
Example:

Laptop or PC IP-address: 192.168.0.120

Arduino IP-address: 192.168.0.100 Port: 8888



Select 'Connect TCP' to open the TCP settings window.



Insert 'IP-address and port number' + confirm.

**Keep in mind to use the IP-address format of 12 numbers!**

IP-address: **000.000.000.000** (12 numbers + ENTER)

Port number: 00000 (5 numbers + ENTER)

Example: 192.168.1.15 Port: 80

IP-address: 192.168.001.015

Port number: 00080



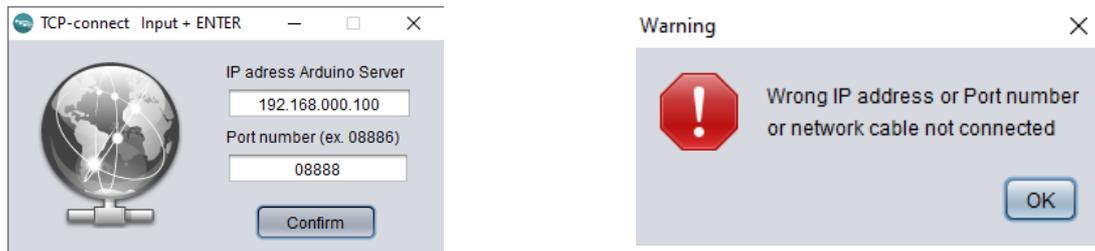
Always press enter when you have entered the field.

If everything is filled in, the IP address and port number will appear in the top bar, the border color of the "TCP button" is green (connection ON).

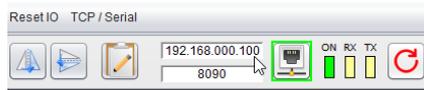


## Arduino IO Simulator Drag & Draw

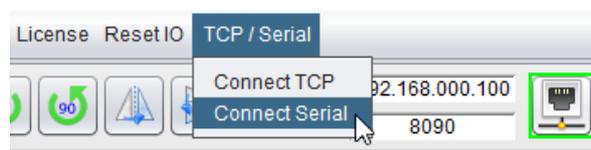
If you fill in a wrong IP-address or port number or the network cable is not connected then after 20s a warning window appears.



If the IP-address is accepted and you want to change it again, then you can only change it by clicking on the IP-address or port field, a input window appears to fill it in.



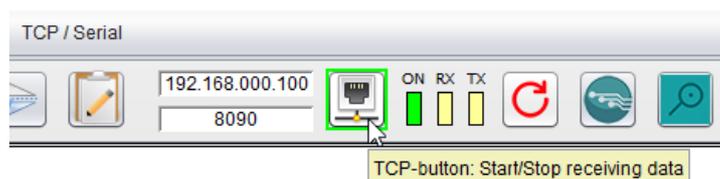
With the menu field is it then only possible to change between serial or TCP. The reason for this is to switch quickly between serial and TCP.



## TCP CONNECTION BUTTON

With the TCP button we can start and stop the receiving data, this button also helps to re-establish the TCP connection if we have lost it.

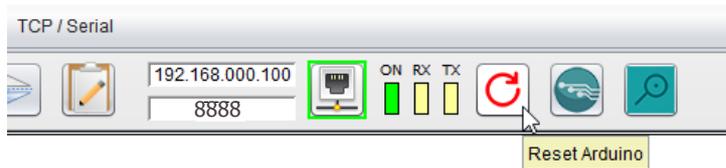
After a manual reset of the Arduino, we have to stop and restart with this button for reconnecting the simulator with the Arduino.



## RESET ARDUINO BUTTON

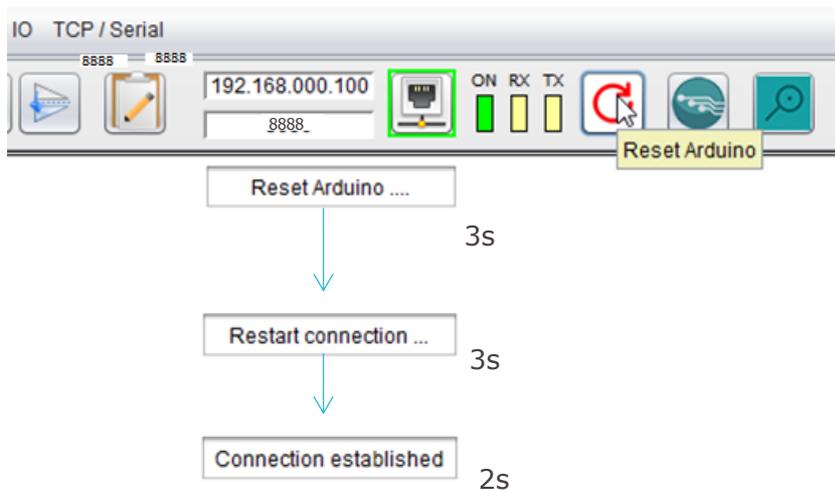
You may only use this button if the connection has been established.

With this button you restart the Arduino (reset Arduino), you need this to give the simulator all the starting data of the Arduino program. (example: Parking\_TCP.ino)

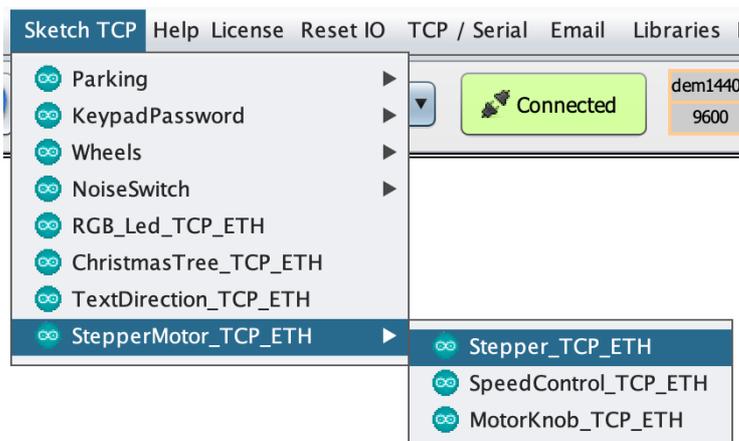


We can follow the Arduino Reset status through a text field that shows us the different steps.

A reset takes approximately 8 seconds.

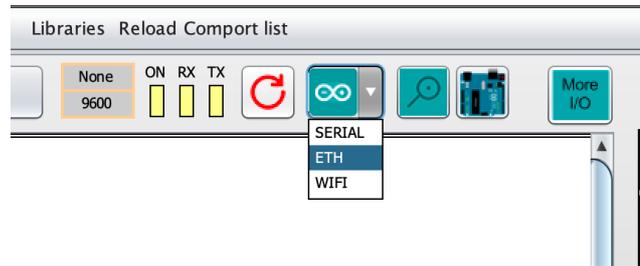


There are 8 sketch examples that work with a TCP ethernet shield. 4 of them are sketches that also work with a Arduino UNO WIFI rev2 or something else with WIFI functionality.



## NEW SKETCH

Select the item 'ETH' to open a new ethernet Arduino sketch:



Here you find the Arduino code for making a TCP connection between the \* lines (\*\*\*)  
For your sketch, you must change the IP address, port name, gateway and subnet.

```

ArduinoUNOSimulatorTCPETHSoftware SimulatorTCPETHprogram.h
//***** code for SimulatorTCP *****
#include <SPI.h> // Library for ethernet
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
int portname = 8888; // Insert server portname
byte ip[] = { 192, 168, 0, 168 }; // Insert arduino server ip adress
byte gateway[] = { 192, 168, 0, 1 }; // internet access via router
byte subnet[] = { 255, 255, 255, 0 }; // subnet mask
EthernetServer server(portname);

#include "SimulatorTCPETHprogram.h" // ArduinoSimulatorTCP library
//***** end code SimulatorTCP *****

// your declaracing code
    
```

All the TCP-libraries needs a portname for sent the data to the simulator, in comment can you see the port names, delete the `//` for the library you need.

```

void setup() {

//***** code for SimulatorTCP *****
Serial.begin(9600); // Simulator Serial Connection code
inString.reserve(10);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB port only
}
Ethernet.begin(mac, ip, gateway, subnet);
server.begin(); // Start the server.
Serial.print("Init");
delay(5000);
//lcd.port(portname); // The lcd library needs this portname
//myservo.port(portname); // The servo library needs this portname
//stepper.port(portname); // The stepper library needs this portname
//irrecv.port(portname); // The IRremote library needs this portname
//irsend.port(portname); // The IRremote library needs this portname

//***** end code SimulatorTCP *****

// *** your setup code ***

}
    
```

Here you find the Arduino simulation code in the void loop().

The function 'TCP\_Recv()' is very important and must be on the first line.

You may have to use this function "TCP\_Recv ()" several times in your program to read the data at the right times:

- After a delay()
- In or after a for next loop
- In or after a doWhile()

```
void loop() { |
    TCP_Recv();           // Put this funtion always on the first line
                        // After a delay()
                        // In a for next loop
                        // In a do while loop

    // *** your programming code ***
}
```

Example:

```
void loop() {
    TCP_Recv(); // Put this funtion always on the first line
                // After a delay()
                // In a for next loop
                // In a do while loop

    sensorValue = analogRead(sensorPin);           // Reading sensor value of the Simulator
    PotValue_Gain = analogRead(Potmeter_Gain);
    PotValue_Dim = analogRead(Potmeter_Dim);
    DimWaarde = (1023 - PotValue_Dim)/4;           // 0V potmeter_Dim (PotValue_Dim=0) => DimWaarde = 255 => ledstrip max
                                                // 5V potmeter_Dim (PotValue_Dim=1023) => DimWaarde = 0 => ledstrip min

    teller = 0;

    if (setLed == 1) {                             // ledstrip on
        analogWrite(Bright,DimWaarde);             // Write to Analog output (0V <-> 5V) Arduino Simulator part
    }

    if ((sensorValue > PotValue_Gain) && (setLed == 0)) { // Geluidswaarde > grenswaarde en ledstrip uit = Ledstrip aan
        digitalWrite(ledPin, HIGH);               // Control led on
        setLed = 1;
        analogWrite(Bright,DimWaarde);             // Write to Analog output (0V <-> 5V) Arduino Simulator part
        delay (2000);                               // 2s wachten
    }

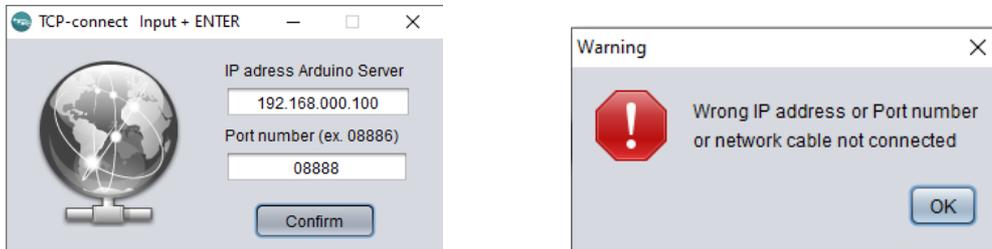
    TCP_Recv();
    sensorValue = analogRead(sensorPin);           // reading microphone signal
    PotValue_Gain = analogRead(Potmeter_Gain);     // reading slidersignal Gain

    if ((sensorValue > PotValue_Gain) && (setLed == 1)) { // Geluidswaarde > grenswaarde en ledstrip aan = Ledstrip uit
        digitalWrite(ledPin, LOW);                 // Arduino Simulator code
        setLed = 0;
        for (teller=0; teller <= DimWaarde;) {
            analogWrite(Bright,DimWaarde-teller); // writing to Analog output
            teller = teller + 5;
            delay(20);
        }
        delay (2000);
    }
}
```

## WARNINGS AND ERRORS

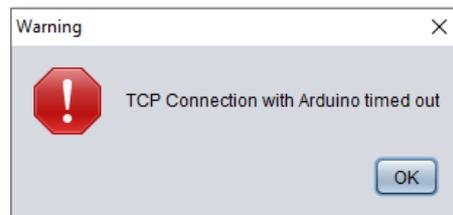
If there is a problem with the TCP connection between the simulator and the Arduino then we can get the following warnings.

If the network cable is not connected after confirm, this window appears after 20 seconds.



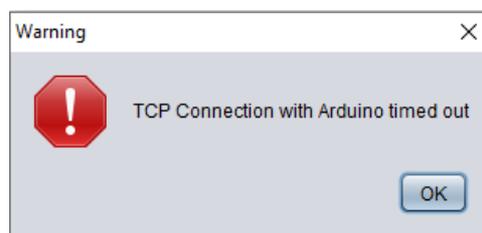
If we don't have any connection when pressing the button 'Reset Arduino', this window appears after 10 seconds.

First solve the connection problem and get the connection back in order by pressing the 'TCP button' off/on.



By quickly pressing the 'TCP button off/on' several times in succession then it is possible that we lose the connection, a window appears after 20 seconds.

To get the connection back in order, we have to manually reset the Arduino and then pressing the TCP-button off/on or restart the simulator program.



## TCP CONNECTION ON A WIDE AREA NETWORK

to get the Arduino on the global network you must perform a port forwarding in the modem.

### Example

Ip address of the modem where Arduino is connected: 091.250.356.028

**Arduino** Forwarded ip-address: 192.168.1.105 Port:8888

**Simulatorprogram** at home input IP/port: 091.250.356.028 Port:8888

For Arduino side: search your own ip-address, you can do this as follow:

<https://whatismyipaddress.com> (ex. 091.250.356.028) or take a look in your modem setup 192.168.1.1

With port forwarding you can open a port in your modem for connect with the Arduino.

Example

**TCP port:** 8888

**forwarded port:** 192.168.1.105

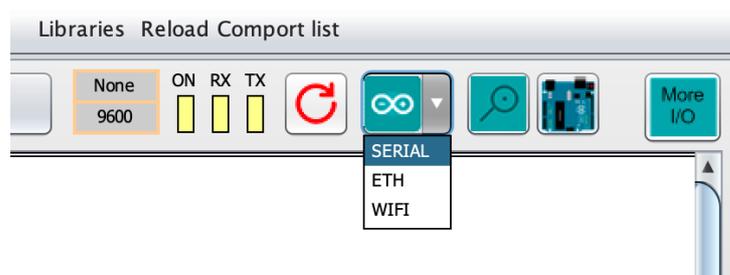
In the simulator write the Arduino worldwide IP address with port number + click confirm.



The Arduino code is as follow:

```
int portname = 8888; // Insert server portname
byte ip[] = { 192, 168, 1, 105 }; // Insert arduino server ip address
byte gateway[] = { 192, 168, 1, 1 }; // internet access via router
byte subnet[] = { 255, 255, 255, 0 }; // subnet mask
```

Use the dropdown selection box to open a new sketch for a Serial, Ethernet or WIFI.



## WIRELESS CONNECTION WITH ARDUINO WIFI

With an Arduino WIFI board makes a wireless connection to your network. As a result, the Arduino TCP ethernet sketch and library will not work. Therefore we have made some examples that work with the WIFI network library.

```
//***** code for SimulatorTCP *****
#include <SPI.h> // Library for ethernet
#include <WiFi.h>

char ssid[] = "network-name"; // Network name to connect to
char pass[] = "network password"; // Network password
int keyIndex = 0;

int status = WL_IDLE_STATUS;
WiFiServer server(80);

#include "SimulatorTCPWiFiProgram.h" // ArduinoSimulatorTCP library
//***** end code SimulatorTCP *****
```

Arduino TCP with ethernet connection code with forwarded port 192.168.1.105:8888

```
ArduinoUNOSimulatorTCPETHSoftware | Arduino 1.8.10
Bestand Bewerken Schets Hulpmiddelen Help

ArduinoUNOSimulatorTCPETHSoftware $ SimulatorTCPETHprogram.h

//***** code for SimulatorTCP *****
#include <SPI.h> // Library for ethernet
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
int portname = 8888; // Insert server portname
byte ip[] = { 192, 168, 1, 105 }; // Insert arduino server ip address
byte gateway[] = { 192, 168, 1, 1 }; // internet access via router
byte subnet[] = { 255, 255, 255, 0 }; // subnet mask
EthernetServer server(portname);

#include "SimulatorTCPETHProgram.h" // ArduinoSimulatorTCP library
//***** end code SimulatorTCP *****

// your declarating code

void setup() {

//***** code for SimulatorTCP *****
Serial.begin(9600); // Simulator Serial Connection code
inString.reserve(10);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB port only
}
Ethernet.begin(mac, ip, gateway, subnet);
server.begin(); // Start the server.
Serial.print("Init");
delay(5000);
//***** end code SimulatorTCP *****

// your setup code

}

void loop() {
TCP_Recv(); // Put this funtion always on the first line

// your programming code

}
```

## PORT FORWARD ON MODEM B-BOX3

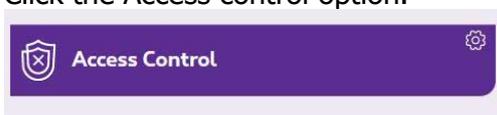
Port opening or forwarding (NAT) is sometimes requested by certain software and peripherals such as video games consoles (Nintendo, PlayStation, Microsoft Xbox, etc.), home automation equipment, etc.

Log in to your modem at home through the address [192.168.1.1](http://192.168.1.1)

Identify yourself on the modem with the user password provided (located under or on the rear of the modem).



Click the Access control option.



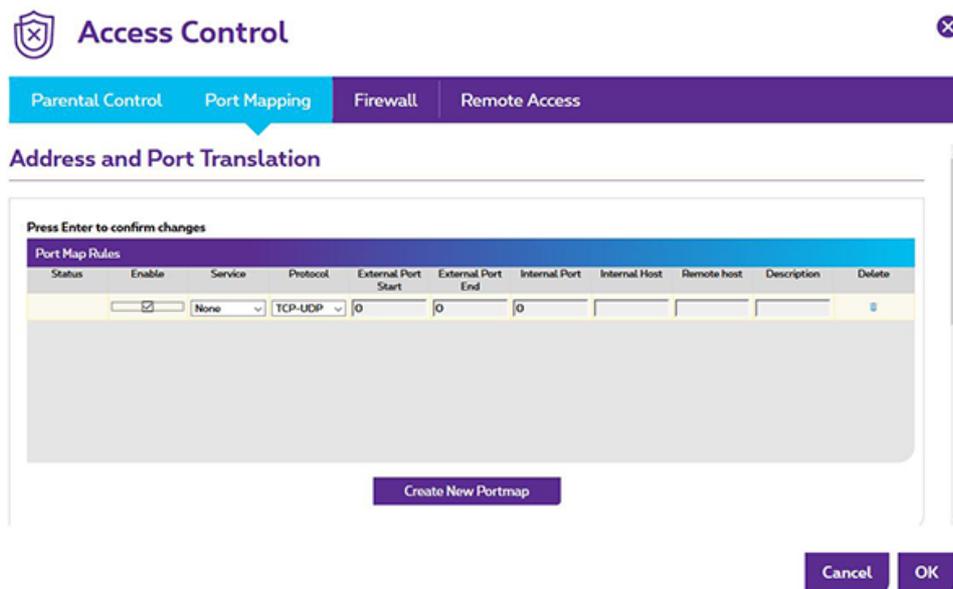
Select the Port mapping tab.

Click the Create new portmap button.

Select the required protocol (TCP for example) and enter the desired port number.

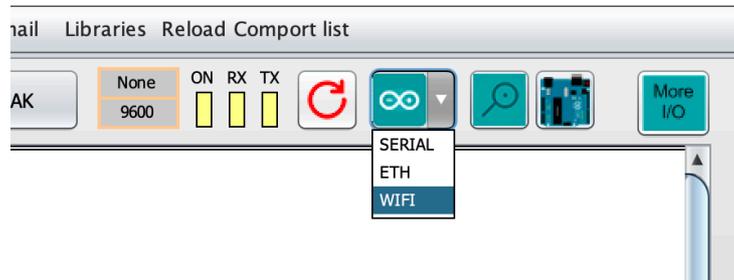
Enter the IP address of the peripheral that requires port forwarding.

Click OK to save the settings.



## ARDUINO SIMULATOR SKETCH

Open a new Arduino IO Simulator sketch by selecting one of the 3 options in the dropdown menu (the sketches can't be saved, save it with 'save as...').



- Start the 'Arduino IDE' application



Now you can set your own code into the Arduino, if it's uploaded in the Arduino you can test it with the Simulator by connecting the simulator after uploading.

- Uploading of a new program to the Arduino UNO
- Start the Arduino IDE
- Open the sketch
- Connect the Arduino UNO board with the computer:



- Select board 'Arduino UNO'
- Select the Serial port
- Upload the program into the Arduino UNO



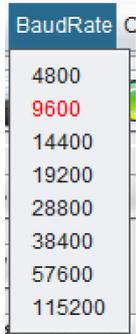
**Attention:** The BaudRate on the simulator is default set on **9600**.

**Make sure you disconnect the simulator before uploading the Arduino code.**

# CONFIGURE THE SERIAL PORT

## Set the BaudRate

The BaudRate is set by default at 9600 or change the BaudRate in the Arduino code and in the Simulator.



## Select the Com port

First you need to select the serial port in the dropdown list, the USB port that is used by the Arduino. The 'Reload Comport list' button in the menu bar will reload the list.

Before the selection

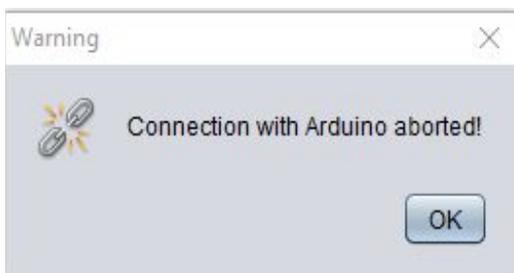


After the selection



## Attention:

You will get a error message when you want to connect with a pin that is already used. If you unplug your Arduino board when you are connected to the Simulator you get a warning notification.

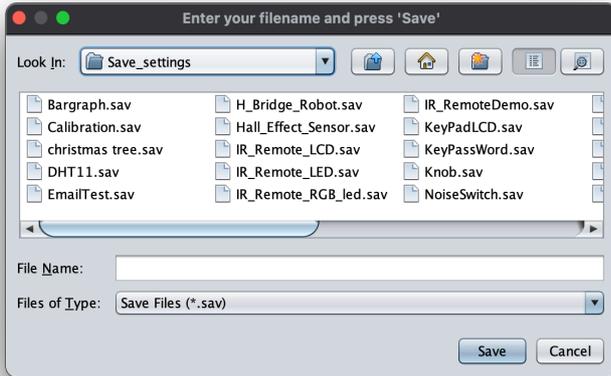


## SAVE AND RESTORE OF SETTINGS

With 'Save' you can save your selected I/O, dictated texts and drawings.  
the 'Restore' button restores the settings to make it easy to use.

We can save the filename of the extension with \*.sav or \*.txt.  
You find the 'Save and Restore' function under 'File'.

### Save



### Restore



Before closing the Simulator you will be asked if you want to save your work.



## SERIAL MONITOR

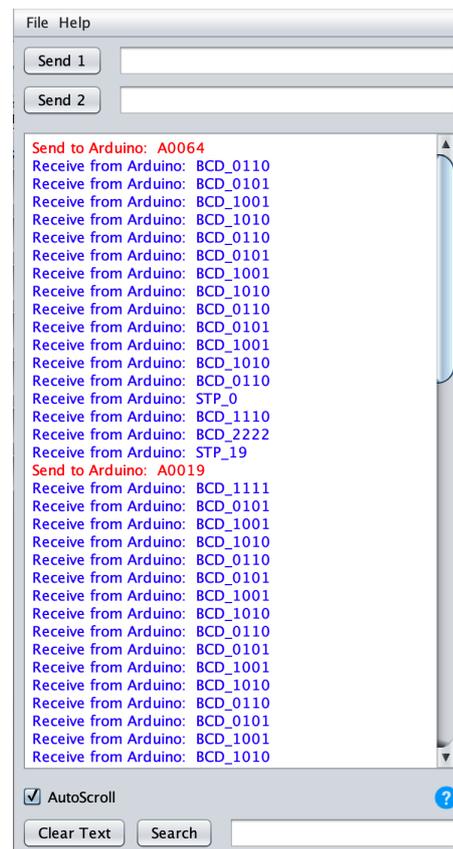
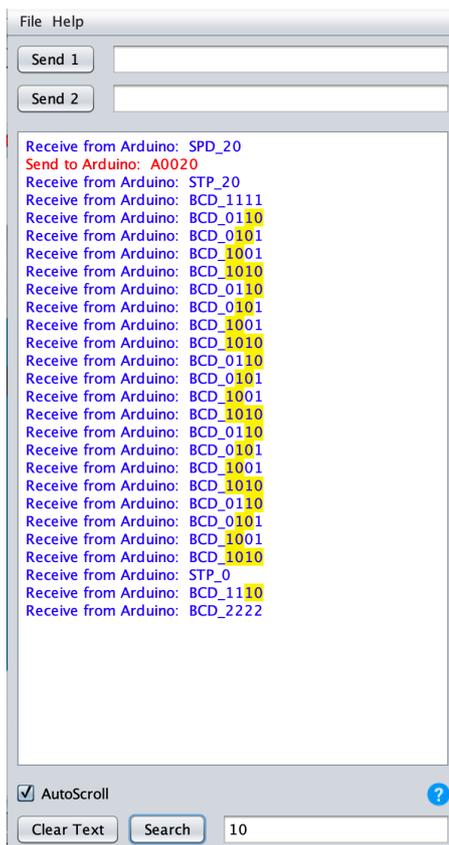
In the simulator, it's now also possible to view the serial data. There are 2 buttons that you can use to send a signal to the Arduino. The monitor shows you a 'Receive from Arduino' line when the Arduino sends data to the simulator and a 'Send to Arduino' line will be shown when you send something to the Arduino with the simulator.

If you want to save the serial monitor output you can save the whole text or a selected area.

With the search function, it's possible to search for a specific word or character. if the word or character is found, it will be highlighted in yellow. By clicking on the blue question mark you get the explanation of all the simulator codes.

By using Serialprint(); in the Arduino IDE, you can send a serial message.

The serial message will start with 'txt\_' in the serial monitor.



## EMAIL LOGGER

**You must have a Google account for this item.**

If you don't have a google account, then go to Google and make an account, this is for free

**To create a Google account:**

1. Go to [www.google.com](http://www.google.com). ...
2. Click **Create** an **account**.
3. The signup form will appear. ...
4. Review **Google's** Terms of Service and Privacy Policy, click the checkbox, then click Next step.
5. The **Create** your profile page will appear. ...
6. Your **account** will be created, and the **Google** welcome page will appear.

How to make a google account: <https://edu.gcfglobal.org/en/googleaccount/creating-a-google-account/1/>

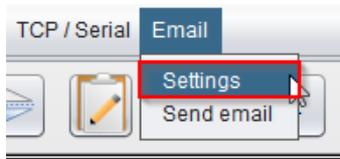
**Important:**

Gmail considers regular email programs and backup programs (i.e. Java Programs , Outlook, etc.) to be "less secure", so in order for them to get access into your account, your "**Allow less secure apps**" option must be turned on.

**Turning on 'less secure apps' settings as mailbox user**

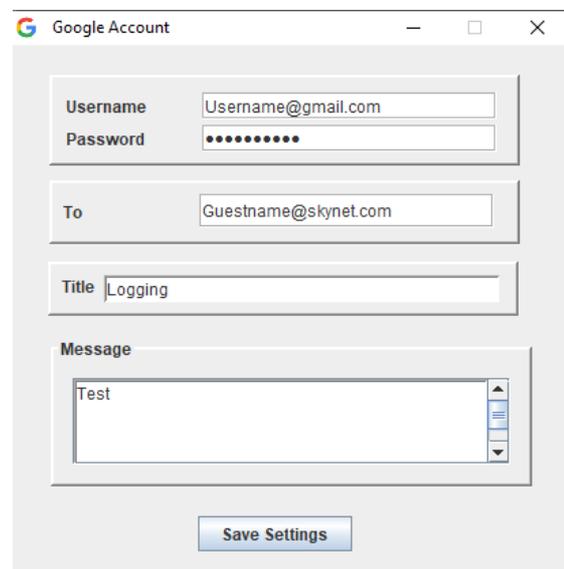
1. Go to your ([Google Account](#)).
2. On the left navigation panel, click **Security**.
3. On the bottom of the page, in the Less secure app access panel, click **Turn on access**.  
**If you don't see this setting, your administrator might have turned off less secure app account access (check the instruction above).**
4. Click the **Save** button.

Go to the settings in the simulator to setup the username and password for sending emails.

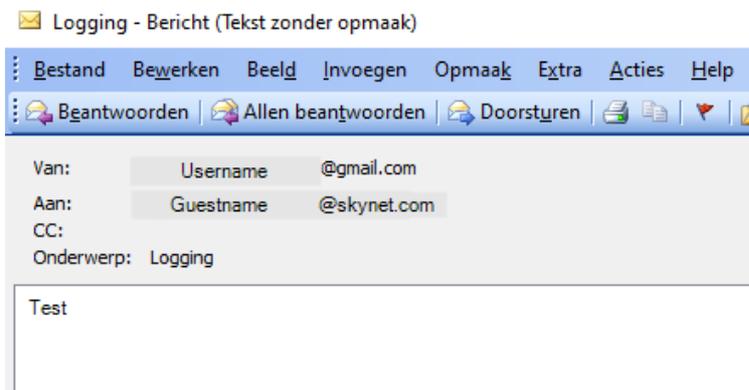


Enter your own google Username and password + Enter the guest name to receive the mail.

You can enter a title and a message. After this you save the settings. Now you can send the email by pressing the "Send email" button.



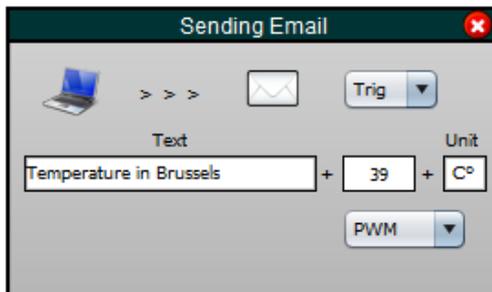
This message appears in the inbox mail from the guest:



## THE EMAIL BOX

With the 'Trig' combobox you can select a trigger output in the range from D2 to D13. With the 'PWM' combobox you can select a output value in the range from 'pwm3 to pwm11. Enter a text and unit, for example Temperature ... C°.

The field with the 39 in will show the value of the selected trigger output.



Example sketch 'Emailtest': send temperature to the Guestname@skynet.com when pressing the button.

With the slider 'A0' you can simulate the temperature (Arduino receive the temp value).

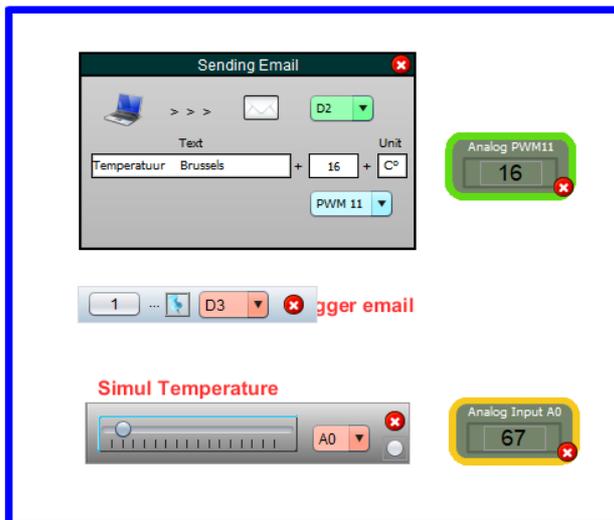
Select output 'D2' for the trigger (Arduino triggers the output).

Select output 'PWM11' for sending temperature value (Arduino send it to the simulator).

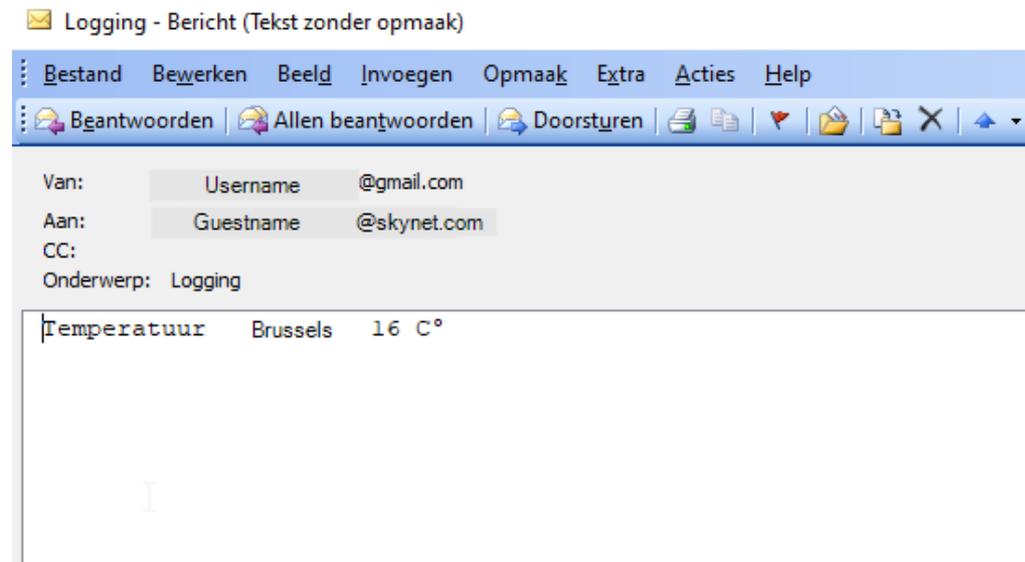
Select input 'D3' for the button (Arduino receive commando)

By pressing the button you will send an email to the guest name (email address).

### Sending temperature



This message appears in the inbox from the guest:



## CHECK FOR UPDATES

You can check if there are any updates available for the Arduino IO Simulator by clicking file → Check Updates. This will launch a window that checks for updates and will give you a link to the download page. Just replace the simulator application to install the newest version.



## SOURCES

Bakker, B. d. (sd). *MAX7219 LED dot matrix display Arduino tutorial*. (Makerguides)  
Opgeroepen op Augustus 4, 2021, van <https://www.makerguides.com/max7219-led-dot-matrix-display-arduino-tutorial/>

marco\_c. (2018, April 19). *Parola A to Z – Sprite Text Effects*. (Arduino plusplus)  
Opgeroepen op Augustus 4, 2021, van  
<https://arduinoplusplus.wordpress.com/2018/04/19/parola-a-to-z-sprite-text-effects/>

*Hall effect sensor*. (2021, July 21). (Wikipedia) Opgeroepen op 2021, van  
[https://en.wikipedia.org/wiki/Hall\\_effect\\_sensor](https://en.wikipedia.org/wiki/Hall_effect_sensor)