

B4R Booklet



B4R Example Projects

1	Getting started	6
1.1	Useful links	6
2	Arduino UNO board.....	7
2.1	Power supply	8
2.2	Pins.....	8
2.2.1	Power pins.....	8
2.2.2	Digital Input / Output pins	9
2.2.3	Analog input pins	9
2.3	Input modes INPUT / INPUT_PULLUP	9
2.4	Basic Pin functions.....	10
2.4.1	Initialize.....	10
2.4.2	DigitalRead	11
2.4.3	DigitalWrite.....	11
2.4.4	AnalogRead.....	11
2.4.5	AnalogWrite.....	12
2.5	RunNative inline C.....	13
3	B4R differences versus B4A/B4J/B4i.....	14
3.1	No user interface	14
3.2	Memory	14
3.3	Stack Buffer	15
3.4	ByteConverter	15
3.5	Concatenation.....	16
3.6	String methods	17
3.7	Encoding	18
3.8	Variables	18
3.9	New Keywords.....	19
3.9.1	AddLooper	19
3.9.2	AvailableRAM	19
3.9.3	CallSubPlus	19
3.9.4	Delay	20
3.9.5	DelayMicroseconds.....	20
3.9.6	JoinBytes.....	20
3.9.7	JoinStrings.....	20
3.9.8	MapRange	21
3.9.9	Micros	21
3.9.10	Millis	21
3.9.11	StackBufferUsage	21
3.10	Variable Types	22
3.10.1	Array variables	23
3.10.2	Array of objects.....	24
3.10.3	Type variables	25
3.11	Casting	26
4	First example programs.....	27
4.1	Button.b4r	28
4.1.1	Sketch.....	28
4.1.2	Code	29
4.2	LedGreen.b4r	30
4.2.1	Sketch.....	30
4.2.2	Code	31
4.3	LedGreenNoSwitchBounce.b4r	32
4.3.1	Sketch.....	33
4.3.2	Code	34
5	More advanced programs Arduino Uno.....	35

5.1	TrafficLight.b4r.....	35
5.1.1	Sketch.....	35
5.1.2	Code.....	36
5.2	LightDimmer.b4r.....	38
5.2.1	Sketch.....	38
5.2.2	Code.....	39
5.3	DCMotor.b4r.....	41
5.3.1	Sketch.....	41
5.3.2	Code.....	42
5.4	DCMotorHBRidge.b4r.....	44
5.4.1	Sketch.....	45
5.4.2	Code.....	46
5.5	ServoMotor.b4r.....	48
5.5.1	Sketch.....	48
5.5.2	Code.....	49
5.6	PulseWidthModulation.b4r.....	51
5.6.1	Sketch.....	52
5.6.2	Code.....	53
5.7	PulsePeriodModulation.b4r.....	55
5.7.1	Sketch.....	56
5.7.2	Code.....	57
5.8	DCMotor slow motion.....	59
5.8.1	Sketch.....	59
5.8.2	Code.....	60
5.9	LCDDisplay.b4r.....	63
5.9.1	LCD display.....	65
5.9.2	Sketch.....	66
5.9.3	Code.....	67
5.10	PulseWidthMeter.....	69
5.10.1	Sketch.....	69
5.10.2	Code.....	70
5.11	ObjectArrays.b4r.....	71
5.11.1	Sketch.....	71
5.11.2	Code.....	72
6	HC-05 Bluetooth.....	77
6.1	Program HC05LedOnOff.b4r.....	78
6.1.1	Sketch.....	79
6.1.2	Code.....	80
6.1.2.1	B4R Arduino UNO.....	80
6.1.2.2	B4A Android.....	81
6.2	Program HC05LightDimmer.....	83
6.2.1	Sketch.....	84
6.2.2	Code.....	85
6.2.2.1	B4R Arduino.....	85
6.2.2.2	B4A Android.....	85
6.3	Program HC05DataLogger.b4r.....	86
6.3.1	Sketch.....	87
6.3.2	Code.....	88
6.3.2.1	B4R Arduino.....	88
6.3.2.2	B4A Android.....	89
6.4	Program HC-05 DCMotor.....	91
6.4.1	Sketch.....	92
6.4.2	Code.....	93

6.4.2.1	B4R Arduino	93
6.4.2.2	B4A Android	94
7	HC-SR04 Ultrasonic Range Sensor	95
7.1	HC-SR04 Simple demo project	96
7.1.1	Sketch	96
7.1.2	Code	97
7.1.2.1	B4R Arduino	97
8	ESP8266 / WeMos board D1 R2	98
8.1	Difference in pin assignment Ardiono Uno > WeMos	99
8.2	Configuration	100
8.3	ESD_LEDGreen.b4r	103
8.3.1	Sketch	103
8.3.2	Code	104
8.4	WiFi Remote Configutation	105
9	FAQ	106
9.1	"Please save project first" message	106
9.2	"Are you missing a library reference" message	106
9.3	How loading / updating a library	107
9.4	Split a long line into two or more lines	107
9.5	"Process has timeout" message	108
9.6	How to pass an Array to a Sub	109
9.7	Select True / Case trick	109
10	Glossary	110
10.1	Electricity basics	110
10.2	PWM Pulse Width Modulation	110

Main contributors: Klaus Christl (klaus), Erel Uziel (Erel)

To search for a given word or sentence use the Search function in the Edit menu.

All the source code and files needed for the example projects in this guide are included in the SourceCode folder.

Covers B4R version 4.00.

Material:

- [Arduino Starter Kit](#)
- HC-05 Bluetooth module
- ESP8266 / WeMos board D1 R2 board

The diagrams for the projects were realized with the [Fritzing](#) software.

[B4X Booklets:](#)

B4X Getting Started

B4X B4X Language

B4X IDE Integrated Development Environment

B4X Visual Designer

B4X Help tools

B4XPages Cross-platform projects

B4X CustomViews

B4X Graphics

B4X XUI B4X User Interface

B4X SQLite Database

B4X JavaObject NativeObject

B4R Example Projects

1 Getting started

B4R - The simplest way to develop native, powerful Arduino programs.

B4R follows the same concepts of the other B4X tools (B4A, B4i, B4J), providing a simple and powerful development tool.

Compiled apps run on Arduino compatible boards.

This guide covers some more advanced topics with example projects.

Basic information on B4R is explained in the [B4X booklets](#) below:

- [B4X Getting started](#)
- [B4X language](#)
- [B4X IDE Integrated Development Environment](#)

1.1 Useful links

Useful links:

Information about the Arduino UNO in the official Arduino site.

[Arduino UNO](#)

Arduino SDK Language reference.

[Arduino Language Reference](#).

2 Arduino UNO board

In this chapter I will explain some basic functions of the Arduino UNO board which may be useful as a reminder.

The Arduino UNO board is the basic board of the Arduino family.

There exist other more advanced boards.

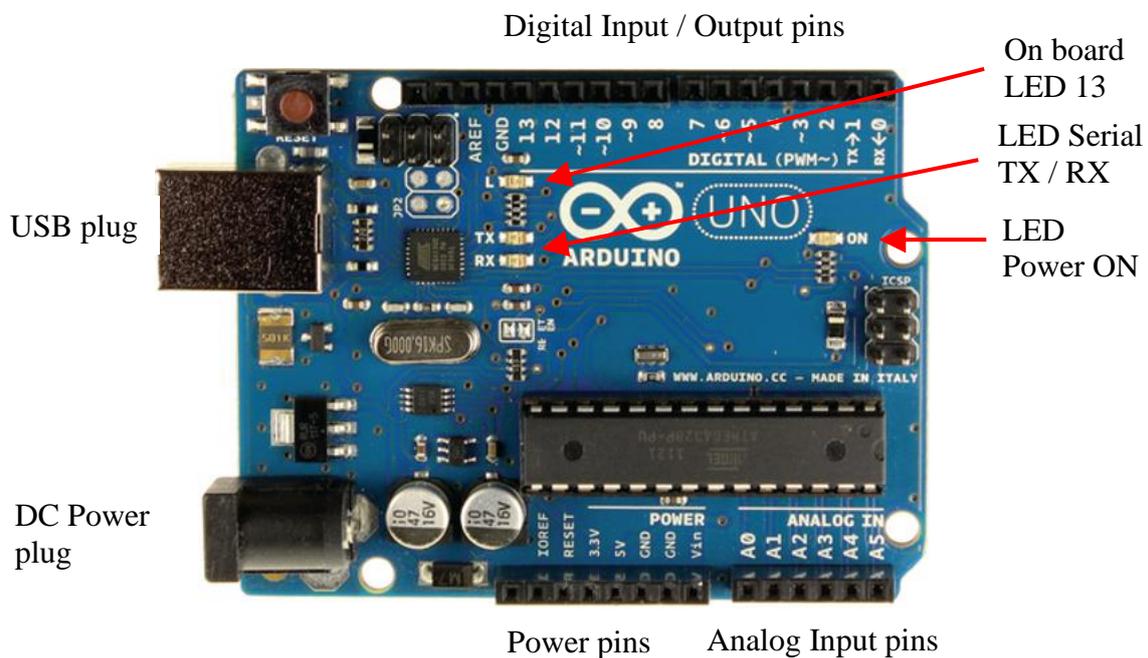
- Arduino DUE
- Arduino MEGA
- Arduino MICRO
- etc. see [Compare board specs](#).

Additional boards called ‘Shields’ can be clipped onto the Arduino boards.

- Arduino Wi-Fi Shield 101
- Arduino Ethernet Shield
- etc.

The Arduino UNO:

The source of the information in this chapter is a summary from the [Arduino site](#).



2.1 Power supply

The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V).

Supplying voltage via the 5V or 3.3V pins bypasses the regulator and can damage your board (see the pins below). We do not advise it.

2.2 Pins

The Arduino UNO has 3 pin sockets:

- Power pins.
- Digital Input / Output pins.
- Analog Input pins.

2.2.1 Power pins

The Power pins are:

- **GND** Power ground, 2 pins.
- **VIN** Power supply input.

The input voltage to the Uno board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Voltage 7 – 12 V.

- **5V** 5 Volt reference voltage. **Don't provide the power to this pin!**
This pin outputs a regulated 5V from the regulator on the board.
- **3.3V** 3.3 Volt reference voltage. **Don't provide the power to this pin!**
A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **RESET**
Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.
- **IOREF**
This pin on the Uno board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.

2.2.2 Digital Input / Output pins

Each of the 14 digital pins on the Uno can be used as an input or output, using the *pinMode* method, *DigitalRead* and *DigitalWrite* functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.

In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- PWM: ~3, ~5, ~6, ~9, ~10, and ~11. These numbers have this “~” prefix.

They can provide 8-bit PWM ([Pulse Width Modulation](#)) outputs with the *AnalogWrite* function allowing to modulate the brightness of a LED (**L**ight **E**mitting **D**iode) or run DC motors at different speeds.

A value of 0 means always OFF and 255 means always ON.

Usage:

```
pinTest3.AnalogWrite(Value As UInt)
pinTest3.AnalogWrite(196)
```

After *AnalogWrite* the pin will generate a steady square wave of the specified duty cycle until the next call to *AnalogWrite* (or a call to *DigitalRead* or *DigitalWrite* on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

- LED 13: There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

2.2.3 Analog input pins

The Arduino UNO has 6 Analog input pins A0 to A5 with 10 bit analog to digital converters (0 to 1023 resolution).

The reference voltage is 5 Volt allowing a resolution of 4.9 mV per unit.

While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general-purpose input/output (GPIO) pins (the same as digital pins 0 - 13).

2.3 Input modes INPUT / INPUT_PULLUP

If you have your pin configured as an INPUT, and are reading a switch, when the switch is in the open state the input pin will be "floating", resulting in unpredictable results. In order to assure a proper reading when the switch is open, a pull-up or pull-down resistor must be used. The purpose of this resistor is to pull the pin to a known state when the switch is open. A 10 K ohm resistor is usually chosen, as it is a low enough value to reliably prevent a floating input, and at the same time a high enough value to not draw too much current when the switch is closed.

The INPUT_PULLUP mode adds an internal pull up resistor no need to add one externally.

With a pull up resistor the pin returns False when the switch is closed because it sets the input to 0 Volt.

2.4 Basic Pin functions

2.4.1 Initialize

Initializes a pin.

Pin.Initialize(Pin As Byte, Mode As Byte)

Pin is the pin number.

- 0, 1, 2, 3 etc. for digital pins
- Pin.A0, Pin.A1 , Pin.A2 etc. for analog pins.

Mode is one of the three connection modes:

- MODE_INPUT
- MODE_INPUT_PULLUP adds an internal pull up resistor.
- MODE_OUTPUT

Example1: Initialize digital pin 3 as input.

```
Private pinTest1 As Pin  
pinTest1.Initialize(3, pinTest1.MODE_INPUT)
```

Example2: Initialize digital pin 3 as input with pull up resistor.

```
Private pinTest2 As Pin  
pinTest2.Initialize(3, pinTest2.MODE_INPUT_PULLUP)
```

Example3: Initialize digital pin 3 as output.

```
Private pinTest3 As Pin  
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

Example4: Initialize analog pin 3 as input.

```
Private pinTest4 As Pin  
pinTest4.Initialize(pinTest4.A4, pinTest4.MODE_INPUT)
```

The analog pins, on the Arduino UNO, can also be accessed with numbers, like:

```
pinTest4.Initialize(18, pinTest4.MODE_INPUT)
```

Pin.A0 = 14

Pin.A1 = 15

Pin.A2 = 16

Pin.A3 = 17

Pin.A4 = 18

Pin.A5 = 19

Initializing an analog pin as output works like a digital output pin.

2.4.2 DigitalRead

Reads the current digital value of a pin.
The return value is True or False.

Pin.DigitalRead returns a Boolean.

There are two input modes depending on the input signal.

- Pin.MODE_INPUT
- Pin. MODE_INPUT_PULLUP adds an internal pullup resistor for use with a switch.

Example:

```
Private pinTest1 As Pin
pinTest1.Initialize(3, pinTest.MODE_INPUT)
```

```
Private Value As Boolean
Value = pinTest1.DigitalRead
```

The Arduino uses internally 0 and 1 for a boolean variable.

```
Log("State: ", Value)
```

Will write either 0 for False or 1 for True in the Logs. In the code you can use False and True.

2.4.3 DigitalWrite

Writes a Boolean value to the given pin.
It can be used for all digital pins and all analog pins.

Pin.DigitalWrite (Value As Boolean)

Example:

```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

```
pinTest3.DigitalWrite(True) directly with the value.
```

```
pinTest3.DigitalWrite(Value) with a variable.
```

2.4.4 AnalogRead

AnalogRead reads the current value of an analog pin.
The return value is an UInt with values between 0 and 1023 (10 bits).
The reference voltage is 5V.

Example:

```
Private pinPot As Pin
pinPot.Initialize(pinPot.A4, pinPot.MODE_INPUT)
```

```
Private Value As UInt
Value = pinPot.AnalogRead
```

2.4.5 AnalogWrite

AnalogWrite writes a Byte value to the given pin.

AnalogWrite has nothing to do with the analog pins nor with AnalogRead.

AnalogWrite can only be used on the digital pins ~3, ~5, ~6, ~9, ~10, and ~11 on the Arduino UNO, the pins with the ~ prefix.

Pin.AnalogWrite (Value As UInt)

Example: we use digital pin ~3 which allows PWM.

```
Private pinTest3 As Pin  
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

`pinTest3.AnalogWrite(145)` directly with the value.

`pinTest3.AnalogWrite(Value)` with a variable, Value must be a UInt variable.

2.5 RunNative inline C

It is possible to use inline C code and run it with the RunNative method.

Examples:

Set pulse width frequency.

```
Sub SetPulseWidthFrequency
  Private pwm_freq As Int = 100 ' 100hz
  RunNative ("SetPWMFreq", Null)
End Sub
```

```
#If C
void SetPWMFreq(B4R::Object* o)
{
  analogWriteFreq(b4r_main::_pwm_freq);
}
#End If
```

Set AnalogSetWidth function.

```
RunNative ("AnalogSetWidth", Null)
```

```
#if C
void AnalogSetWidth(B4R::Object* o)
{
  analogSetWidth(10);
}
#End if
```

3 B4R differences versus B4A/B4J/B4i

3.1 No user interface

The biggest difference is that there is no user interface in B4R!

3.2 Memory

Memory is a big issue in Arduino.

1. The available RAM is very small compared to other platforms. For example, the Uno has 2k of RAM.
2. There is no sophisticated heap management which means that usage of dynamic memory should be avoided.
3. Programs are expected to run for a long time with zero memory leaks.

There are two types of variables in B4R: local variables and global variables.

Local variables are always created on the stack. This means that any object created in a sub, other than `Process_Globals`, will be destroyed when the sub ends (this is not the case in other B4X tools).

Global objects, which are objects created in `Process_Globals`, are not stored in the stack and will always be available.

Note that global variables, except of primitives, behave as constants. You cannot redim them and they cannot be reassigned.

Consider this code:

```
Sub MySub As Pin
  Dim p As Pin
  Return p
End Sub
```

The compiler will throw the following error:

"Objects cannot be returned from subs (only primitives, strings and arrays of primitives are allowed)".

The reason for this error is that the `Pin` object created in the sub is destroyed when `MySub` ends.

Primitives (numeric types and Booleans) are not problematic as the actual value is passed.

B4R does allow returning strings and arrays of primitives. It involves copying those objects in the stack.

As a general rule, objects should be declared as process global.

3.3 Stack Buffer

B4R maintains an additional "stack" named stack buffer. This buffer is used internally when a method needs to allocate memory. **Remember that the heap is never used.**

For example the NumberFormat keyword accepts a number and returns a formatted string. The string must be allocated somewhere. It is allocated in the stack buffer. Like all other objects it will be destroyed when the sub ends.

The buffer is also when the array size is not known during compilation.

The default size of the stack buffer is 300 bytes. This can be changed with the #StackBufferSize attribute.

If you encounter memory issues then check the value of StackBufferUsage keyword (in the relevant sub).

3.4 ByteConverter

ByteConverter type from the rRandomAccessFile library includes methods to convert objects and other types to bytes and vice versa. It also allows copying objects with the ObjectCopy method. This method can be used to copy a local object to a global variable.

Notes

- Only single dimension arrays are supported.
- Local variables are not set to zero automatically.
You can set the value in the declaration:

```
Dim counter As Int = 0
```

- Arrays are not bound checked

```
Dim i(10) As Int  
i(10) = 100 'bad things can happen as the index of the last element is 9
```

B4R strings are different than in other B4X tools. The reasons for these differences are:

1. Very limited memory.
2. Lack of Unicode encoders.

A String object in B4R is the same as C char* string. It is an array of bytes with an additional zero byte at the end.

The requirement of the last zero byte makes it impossible to create a substring without copying the memory to a new address. **For that reason, arrays of bytes are preferable over Strings.** The various string related methods work with arrays of bytes.

Converting a string to an array of bytes is very simple and doesn't involve any memory copying. The compiler will do it automatically when needed:

```
Dim b() As Byte = "abc" 'equivalent to Dim b() As Byte = "abc".GetBytes
```

3.5 Concatenation

The & operator is not available in B4R. If you do need to concatenate strings or arrays of bytes, then you can use JoinStrings or JoinBytes keywords. They are more efficient as they concatenate all the elements at once.

In most cases you don't need to concatenate strings. A better solution for example when sending strings (or bytes) with AsyncStreams:

```
AStream.Write("The current value is: ").Write(s).Write(" and the other value is: ")
AStream.Write(NumberFormat(d, 0,0))
```

This doesn't work: `Log("State: " & State)`

This works: `Log("State: ", State)`

3.6 String methods

The standard string methods are available in ByteConverter type (rRandomAccessFile library).

They are similar to the string methods in other B4X tools:

```
Private Sub AppStart
    Serial1.Initialize(115200)
    Log("AppStart")
    Dim bc As ByteConverter
    Log("IndexOf: ", bc.IndexOf("0123456", "3")) 'IndexOf: 3
    Dim b() As Byte = " abc,def,ghijkl "
    Log("Substring: ", bc.SubString(b, 3)) 'Substring: c,def,ghijkl
    Log("Trim: ", bc.Trim(b)) 'Trim: abc,def,ghijkl
    For Each s() As Byte In bc.Split(b, ",")
        Log("Split: ", s)
        'Split: abc
        'Split: def
        'Split: ghijkl
    Next
    Dim c As String = JoinStrings(Array As String("Number of millis: ", Millis, CRLF, "Number of micros: ", Micros))
    Log("c = ", c)
    Dim b() As Byte = bc.SubString2(c, 0, 5)
    b(0) = Asc("X")
    Log("b = ", b)
    Log("c = ", c) 'first character will be X
End Sub
```

Note how both strings and array of bytes can be used as the compiler converts strings to arrays of bytes automatically.

With the exception of JoinStrings, none of the above methods make a copy of the original string / bytes.

This means that modifying the returned array as in the last three lines will also modify the original array.

It will also happen with string literals that all share the same memory block:

```
Private Sub AppStart
    Serial1.Initialize(115200)
    Log("AppStart")
    Dim bc As ByteConverter
    Dim b() As Byte = bc.Trim("abcdef ")
    b(0) = Asc("M") 'this line will change the value of the literal string
    dim s as String = "abcdef "
    Log(s) 'Mbcdef
End Sub
```

3.7 Encoding

There is no real support for Unicode encodings in Arduino. You are always working with raw bytes.

```
Dim s As String = "אראל"  
Log(s) 'will print correctly  
Log(s.Length) '8 because each of the non-ASCII characters in this case takes two bytes
```

3.8 Variables

Variable types are different in B4R compared to the other B4x languages. Some variable types don't exist in the other B4x languages like UInt and ULong .

List of types with their ranges:

Numeric types:

Byte 0 - 255
Int (2 bytes) -32768 - 32768. Similar to Short type in other B4x tools.
UInt (2 bytes) 0 - 65535.
Long (4 bytes) -2,147,483,648 - 2,147,483,647. Similar to Int type in other B4x tools.
ULong (4 bytes) 0 - 4,294,967,295
Double (4 bytes) 4 bytes floating point. Similar to Float in other B4x tools.
Float is the same as Double. Short is the same as Int.

The above is true on all boards, including the Arduino Due.

Other types:

Boolean True or False. Practically it is saved as a byte with the value of 1 or 0.

String Strings are made from an array of bytes that end with a null byte (byte with the value of 0).

Object Objects can hold other types of values.

B4R supports only single dimension arrays!

3.9 New Keywords

There are several new Keywords in B4R which do not exist in the other B4x languages.

3.9.1 AddLooper

AddLooper (SubName As SubVoidVoid)

Adds a looper sub.

Looper is like a timer with the lowest possible interval.

There could be any number of looper subs.

Example:

```
AddLooper("Looper1")
```

```
'...
```

```
Sub Looper1
```

```
End Sub
```

Returns: void

3.9.2 AvailableRAM

AvailableRAM

Returns the available memory. This method will only work on AVR chips.

Returns: ULong

3.9.3 CallSubPlus

CallSubPlus (SubName As SubVoidByte, DelayMs As ULong, Tag As Byte)

Runs the given sub after the specified time elapses. Note that this call does not block the thread.

SubName - Sub name as literal string.

DelayMs - Delay in milliseconds.

Tag - Value that will be passed to the called sub. This is useful when multiple CallSubPlus call the same sub.

Example:

```
CallSubPlus("DoSomething", 1000, 5)
```

```
'...
```

```
Sub DoSomething(Tag as Byte)
```

```
End Sub
```

Returns: void

3.9.4 Delay

Delay (DelayMs As ULong)

Pauses the execution for the specified delay measured in milliseconds.
Note that in most cases it is better to use a `Timer` or `CallSubPlus` instead.

Returns: void

3.9.5 DelayMicroseconds

DelayMicroseconds (Delay As UInt)

Pauses the execution for the specified delay measured in microseconds.

Returns: void

3.9.6 JoinBytes

JoinBytes (ArrayOfObjects As Object())

Concatenates the arrays of bytes to a single array.
Think carefully before using this method. In most cases there are better solutions that require less memory (calling `Write` multiple times for example).

Example:

```
Dim b() as byte = JoinBytes(Array("abc".GetBytes, Array as Byte(13, 10)))
```

Returns: Byte()

3.9.7 JoinStrings

JoinStrings (Strings As String())

Concatenates the strings to a single string.
Think carefully before using this method. In most cases there are better solutions that require less memory.

Example:

```
Dim s As String = JoinStrings(Array As String("Pi = ", cPI))
```

Returns: B4RString

3.9.8 MapRange

MapRange (Value As Long, FromLow As Long, FromHigh As Long, ToLow As Long, ToHigh As Long)

Maps the value from the "from" range to the "to" range.

Example:

```
Dim v as Int = MapRange(p.AnalogRead, 0, 1023, 0, 255)
```

Returns: Long

3.9.9 Micros

Returns the number of microseconds since the last restart.

Note that this value overflows (goes back to zero) after approximately 70 minutes.

Returns: ULong

3.9.10 Millis

Returns the number of milliseconds since the last restart.

Returns: ULong

3.9.11 StackBufferUsage

Returns the usage of the stack memory buffer (set with #StackMemoryBuffer attribute).

Returns: UInt

3.10 Variable Types

List of types with their ranges:

Numeric types:

Byte 0 - 255

Int (2 bytes) -32768 - 32768. Like Short type in other B4x tools.

UInt (2 bytes) 0 - 65535. B4R specific.

Long (4 bytes) -2,147,483,648 - 2,147,483,647. Like Int type in other B4x tools.

ULong (4 bytes) 0 - 4,294,967,295 B4R specific.

Double (4 bytes) 4 bytes floating point. Like Float in other B4x tools.

Float is the same as Double. Short is the same as Int.

The above is true on all boards, including the Arduino Due.

Other types:

Boolean True or False. Practically it is saved as a byte with the value of 1 or 0.

String Strings are made from an array of bytes that end with a null byte (byte with the value of 0).

Object Objects can hold other types of values.

Primitive types are always passed by value to other subs or when assigned to other variables.

For example:

Sub S1

Private A As Int

A = 12

S2(A)

Log(A) ' Prints 12

End Sub

The variable A = 12

It's passed by value to routine S2

Variable A still equals 12, even though B was changed in routine S2.

Sub S2(B As Int)

B = 45

End Sub

Variable B = 12

Its value is changed to B = 45

All other types, including arrays of primitive types and strings are categorized as non-primitive types.

When you pass a non-primitive to a sub or when you assign it to a different variable, a copy of the reference is passed.

This means that the data itself isn't duplicated.

It is slightly different than passing by reference as you cannot change the reference of the original variable.

3.10.1 Array variables

Arrays are collections of data or objects that can be selected by indices. Arrays can have multiple dimensions.

The declaration contains the `Private` or the `Public` keyword followed by the variable name `Name`, the number of items between brackets (`5`), the keyword `As` and the variable type `String`.

There does exist the `Dim` keyword, this is maintained for compatibility.

B4R supports only single dimension arrays !

Examples:

```
Public Name(5) As String      One-dimension array of strings, total number of items 5.
Public Pins(5) As Pin        One-dimension array of pins, total number of items 5.
```

The first index of the dimension is 0.

`Name(0)`

The last index is equal to the number of items minus 1.

`Name(4)`

In the other B4x products the declaration below is possible, but **NOT** in B4R.

```
Public NbNames = 10 As Int
Public Name(NbNames) As String
```

It throughs this error:

The size of non-primitive arrays must be a numeric literal.

The size value must be a numeric literal like!

```
Public Name(10) As String
```

Filling an array with the `Array` keyword:

```
Public Days() As String
Days = Array As String("Monday", "Tuesday", "Wednesday", "Thursday" . . . )
```

3.10.2 Array of objects

Objects can also be in an Array. The following code shows an example:

In the example below 4 Pins are used for LEDs and 2 Pins used for pushbuttons. The example is based on the ObjectArrays project.

```

Sub Process_Globals
    Public pinLEDs(4) As Pin
    Public pinButtons(2) As Pin
End Sub

Private Sub AppStart
    Private i As Int

    For i = 0 To 3
        pinLEDs(i).Initialize(i + 7, pinLEDs(i).MODE_OUTPUT)
    Next

    For i = 0 To 1
        pinButtons(i).Initialize( i + 18, pinButtons(i).MODE_INPUT_PULLUP)
        pinButtons(i).AddListener("pinButtons_StateChanged")
    Next

End Sub

Private Sub pinButtons_StateChanged (State As Boolean)
    Private p As Pin

    Log("State: ", State)                'Log the State value

    p = Sender
    Log("Pin Number: ", p.PinNumber)     'Log the Pin Number

    Select p.PinNumber
    Case 18
        'your code for the pushbutton on pin 18 Analog pin A4
    Case 19
        'your code for the pushbutton on pin 19 Analog pin A5
    End Select
End Sub

```

In B4R this code doesn't work:

```

Sub Process_Globals
    Public NbLEDs = 4 As Int
    Public pinLEDs(NbLEDs) As Pin
End Sub

```

It throughs this error:

The size of non-primitive arrays must be a numeric literal.

The size value must be a numeric literal!

3.10.3 Type variables

A Type cannot be private. Once declared it is available everywhere.

The only place to declare them is in the `Process_Globals` routine in the `Main` module.

We will use a Type variable for LEDs.

A type variable is defined with the `Type` keyword followed by a list of variables:

```
Type Leds(PinNumber As Byte, Color As String, State As String)
Public PowerLed As Leds
Public Led(4) As Leds
```

The new personal type is `Leds`, then we declare either single variables or arrays of this personal type.

To access a particular item use following code.

```
Led(0).Color = "Blue"
Led(0).PinNumber = 0
Led(0).State = "ON"
```

The variable name, followed by a dot and the desired parameter.

If the variable is an array, then the name is followed by the desired index between brackets.

It is possible to assign a typed variable to another variable of the same type, as shown below.

```
PowerLed = Led(1)
```

3.11 Casting

B4R casts types automatically as needed. It also converts numbers to strings and vice versa automatically.

In some cases, you need to explicitly cast an Object to a specific type.

This can be done by assigning the Object to a variable of the required type.

For example, the `Sender` keyword references an Object which is the object that raised the event. The following code changes the state of a LED according to the state of a switch.

Note that there are three switches sharing the same event sub.

```
Sub Process_Globals
    Public Serial1 As Serial

    Public pinSwitch(3) As Pin
    Public pinLed(3) As Pin
End Sub

Private Sub AppStart
    Serial1.Initialize(115200)

    Private i As Int
    For i = 0 To 2
        pinSwitch(i).Initialize(i, pinSwitch(i).MODE_INPUT_PULLUP)
        pinSwitch(i).AddListener("pinSwitches")

        pinLed(i).Initialize(1 + 3, pinLed(i).MODE_OUTPUT)
    Next
End Sub

Private Sub pinSwitches_StateChanged (State As Boolean)
    Private Switch As Pin

    Switch = Sender

    pinLed(Switch.PinNumber + 3).DigitalWrite(Switch.DigitalRead)
End Sub
```

4 First example programs

All the projects were realized with the [Arduino Starter Kit](#).

The diagrams for the projects were realized with the [Fritzing](#) software.

When we run the IDE, we get the default code like below.

The #Region Project Attributes is normally collapsed; only the attributes are used with B4R. Leave them as they are.

```
Region Project Attributes
```

```
#AutoFlushLogs: True  
#CheckArrayBounds: True  
#StackBufferSize: 300
```

```
#End Region
```

```
Sub Process_Globals
```

```
'These global variables will be declared once when the application starts.
```

```
'Public variables can be accessed from all modules.
```

```
Public Serial1 As Serial
```

```
End Sub
```

```
Private Sub AppStart
```

```
Serial1.Initialize(115200)
```

```
Log("AppStart")
```

```
End Sub
```

```
Public Serial1 As Serial
```

```
Serial1.Initialize(115200)
```

```
Log("AppStart")
```

Defines the serial interface with the computer.

Initializes the serial port with a Baud rate of 115200 Hertz.

Shows **AppStart** in the [Logs](#) when the program starts.

4.1 Button.b4r

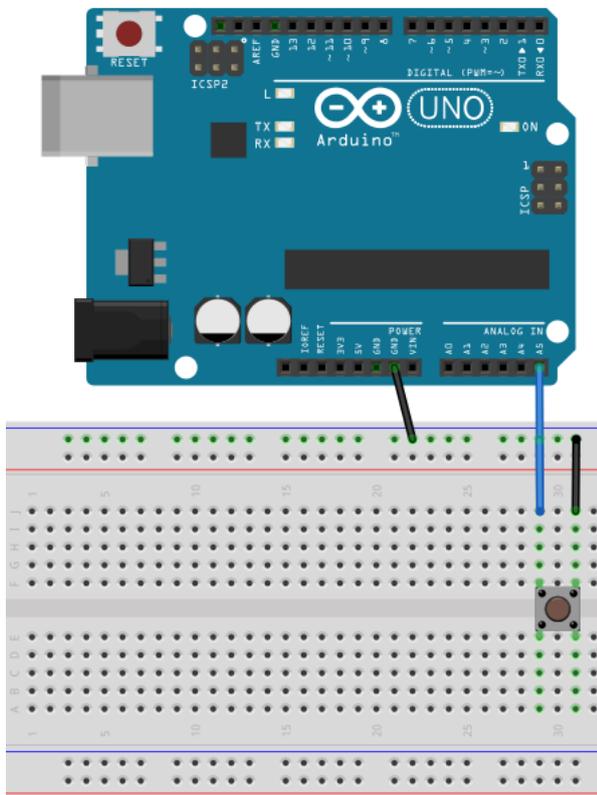
Let us write our first program.

It's like Erels Button example from the forum. It uses a pushbutton switch and the Led 13 on the Arduino UNO board.

The project Button.b4r is available in the SourceCode folder.

- Open B4R.
- Save the project as Button in a folder with the name Button.
- Build the board with the pushbutton and the wires.
- Connect the Arduino to the PC.
- Write the code.
- Run the program.

4.1.1 Sketch



Material:

- 1 pushbutton switch

Connect one Arduino **GND** (ground) pin to the ground **GND** line of the breadboard.

Then connect one pin of the pushbutton switch to the ground line.

And connect the other pin of the pushbutton to pin **A5** of the Arduino analog pins.

We could have connected the first pin of the pushbutton directly to the **GND** pin of the Arduino, but the connection in the image is ready for the next examples.

We could also have used one of the digital pins instead of the analog pin.

4.1.2 Code

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton As Pin      'pin for the button
  Private pinLED13 As Pin      'pin for LED 13 on the Arduino
End Sub
```

We declare the pins for the pushbutton and the on board Led 13.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.AddListener("pinButton_StateChanged")

  pinLED13.Initialize(13, pinLED13.MODE_OUTPUT)
End Sub
```

We initialize pinButton, as the analog pin **A5**, with pinButton.A5 and set the input mode to pinButton.MODE_INPUT_PULLUP. We need a pullup resistor to prevent the pin from floating, MODE_INPUT_PULLUP connects an internal pull up resistor.

We add pinButton.AddListener("pinButton_StateChanged"), to generate a StateChanged event when the state of pin pinButton changes which means that the pushbutton is pressed or released.

We initialize pinLED13, as the onboard Led as digital pin 13 and set the output mode to pinLED13.MODE_OUTPUT.

```
Sub pinButton_StateChanged (State As Boolean)
  Log("State: ", State)
  'state will be False when the button is clicked because of the PULLUP mode.
  pinLED13.DigitalWrite(Not(State))
End Sub
```

We add a Log, Log("State: ", State), to display the state in the Logs.

We write the State to the digital output of the on board led, pinLED13.DigitalWrite(Not(State)).

We write Not(State) because State will be False when the pushbutton is pressed because of the PULLUP mode.

Click on  or press F5 to run the code.

When you press the pushbutton, led 13 on the Arduino UNO will be ON and when you release the pushbutton led 13 will be OFF.

4.2 LedGreen.b4r

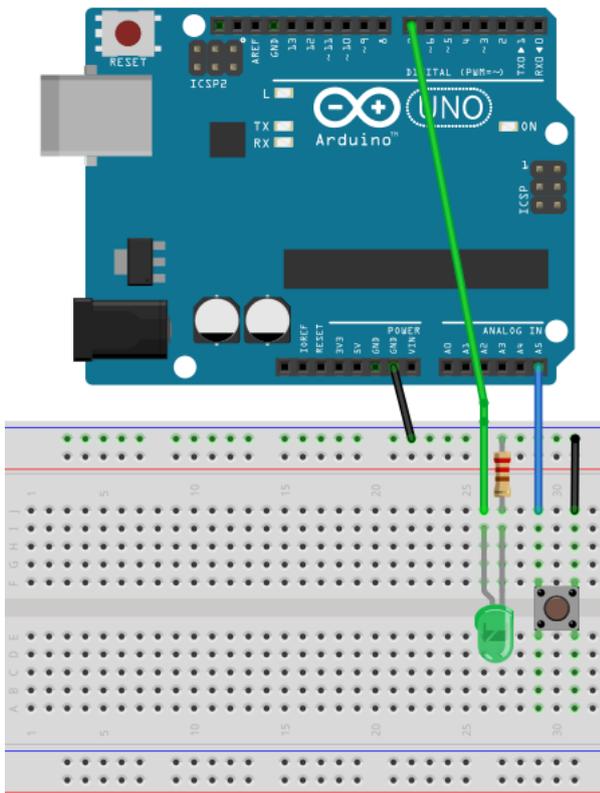
For this project we use a copy of the Button project.

Create a new LedGreen folder, copy the files of the Button project and rename the Button.xxx files to LedGreen.xxx.

We add a green Led which can be switched on and off with the button from the first example.

The project LedGreen.b4r is available in the SourceCode folder.

4.2.1 Sketch



Material:

- 1 pushbutton switch
- 1 green LED
- 1 220 Ω resistor

We keep the mounting of the pushbutton switch from the first example.

One pin on the **GND** line of the breadboard. The other pin to digital pin **7** on the Arduino.

And we

- Add a green Led on the breadboard.
- Connect the cathode (-) via a 220 Ω resistor to the ground **GND** line of the breadboard.
- Connect the anode (+) to digital pin **7**.

4.2.2 Code

```

Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton As Pin           'pin for the button
  Private pinLEDGreen As Pin        'pin for the green Led
  Private LightOn = False As Boolean
End Sub

```

We keep the definition of pinButton.

We change the definition
 Private pinLED13 As Pin
 to
 Private pinLEDGreen As Pin
 for the green Led

We add a global Boolean variable LightOn which is True when the light is ON.

```

Private Sub AppStart
  Serial1.Initialize(115200)

  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.AddListener("pinButton_StateChanged")

  pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
End Sub

```

We leave the code for pinButton.

We initialize pinLEDGreen as digital pin **7** and set the output mode to pinLEDGreen.MODE_OUTPUT.

```

Private Sub pinButton_StateChanged (State As Boolean)
  If State = False Then 'remember, False means button pressed.
    LightOn = Not(LightOn)
    pinLEDGreen.DigitalWrite(LightOn)
  End If
End Sub

```

Every time State is False, pushbutton pressed, we change the variable LightOn and write it to pinLEDGreen.

4.3 LedGreenNoSwitchBounce.b4r

For this project we use the same circuit as LedGreen.b4r.

The only difference is in the code.

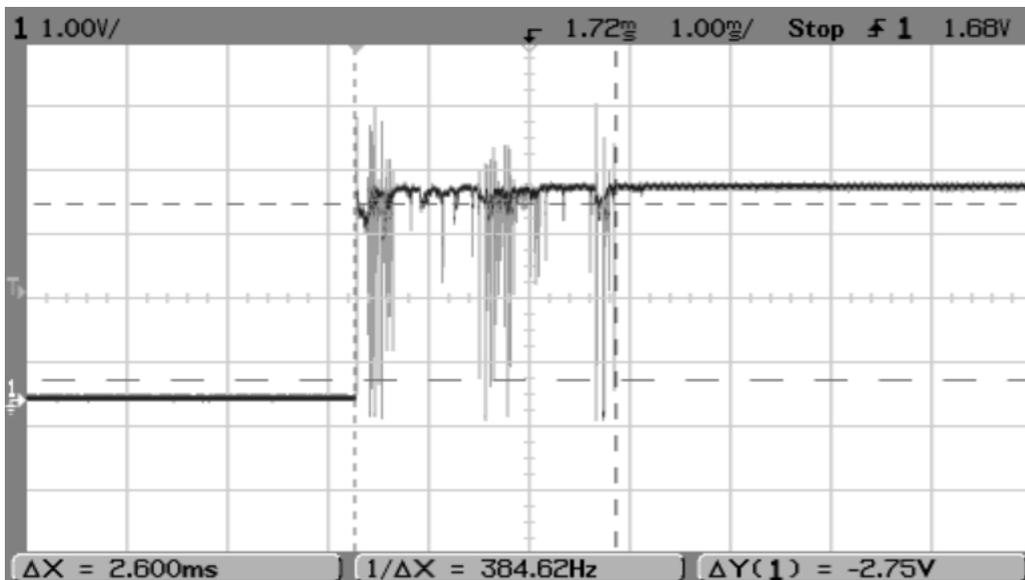
The project LedGreenNoSwitchBounce.b4r is available in the SourceCode folder.

The pushbutton switch we use in our projects has a problem called bouncing.

The signal of a mechanical switch is not clean, the switch has several bounces which are interpreted by the digital inputs as several state changes. If we have an even number of state changes it is similar to having done nothing.

But we want only one state change per button press.

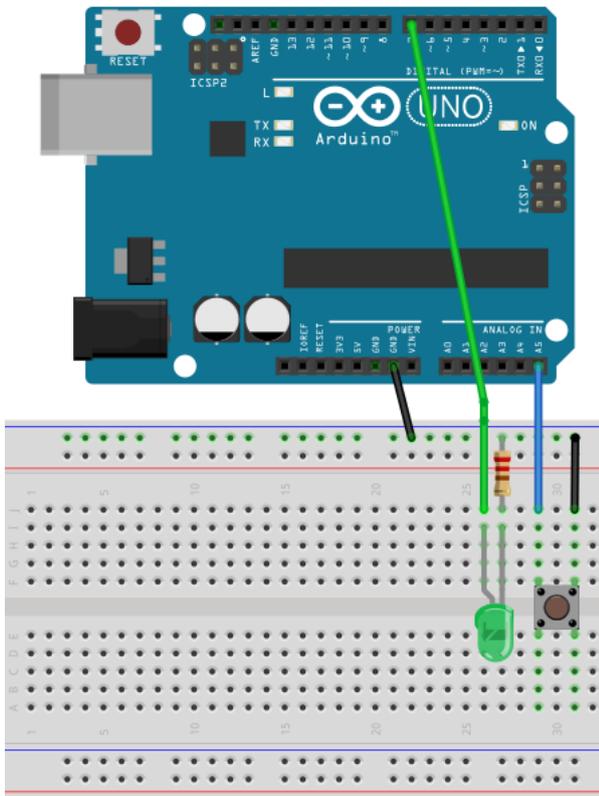
Image of switch bouncing ([source Wikipedia](#)):



The switch bounces between on and off several times before settling.

To solve this problem, we don't react in the `pinButton_StateChanged` routine on state changes within a given time.

4.3.1 Sketch



Material:

- 1 pushbutton switch
- 1 green LED
- 1 220 Ω resistor

We keep the mounting of the pushbutton switch from the first example.

One pin on the **GND** line of the breadboard.
The other pin to digital pin **7** on the Arduino.

And we

- Add a green Led on the breadboard.
- Connect the cathode (-) via a 220 Ω resistor to the ground **GND** line of the breadboard.
- Connect the anode (+) to digital pin **7**.

4.3.2 Code

The code is almost the same as LedGreen.b4r.

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton As Pin           'pin for the button
  Private pinLEDGreen As Pin        'pin for the green Led
  Private LightOn = False As Boolean
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub
```

We add two new variables: BounceTime and BounceDelay.

```
Private Sub AppStart
  Serial1.Initialize(115200)

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
End Sub
```

Same as LedGreen.b4r

New pinButton_StateChanged routine:

```
Private Sub pinButton_StateChanged (State As Boolean)
  If State = False Then 'remember, False means button pressed.
    If Millis - BounceTime < BounceDelay Then
      Return 'Return, bouncing
    Else
      LightOn = Not(LightOn)
      pinLEDGreen.DigitalWrite(LightOn)
      BounceTime = Millis 'reset BounceTime to current time
    End If
  End If
End Sub
```

Every time State is False, pushbutton pressed:

- We check if the time between the current state change and the first state change. If the time is shorter than BounceDelay, which means a bounce, we do nothing. If the time is longer than BounceDelay, which means a real state change, we execute the code.
- We change the variable LightOn and write it to pinLEDGreen.
- Set the new BounceTim

5 More advanced programs Arduino Uno

All the projects were realized with the [Arduino Starter Kit](#).

The diagrams for the projects were realized with the [Fritzing](#) software.

5.1 TrafficLight.b4r

This project is an evolution of the LedGreen project and simulates traffic lights.

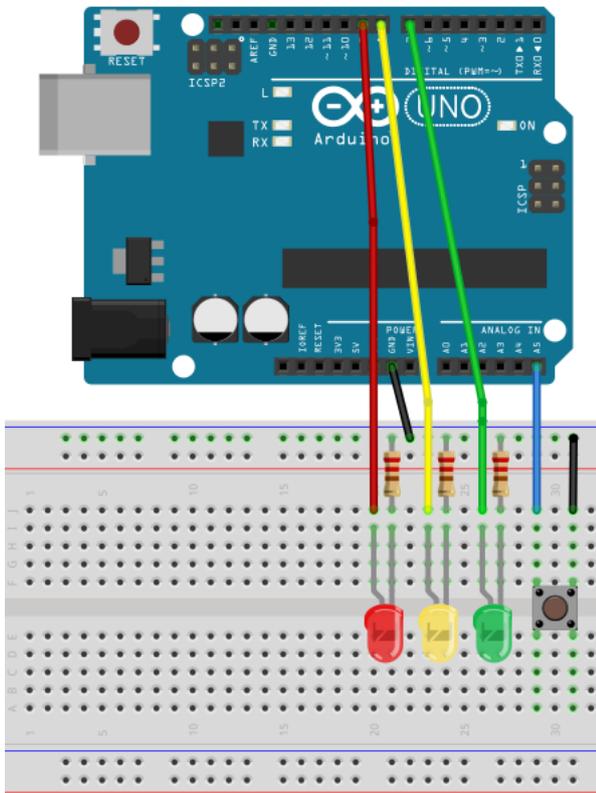
We use a copy of the LedGreen project.

Create a new TrafficLight folder, copy the files of the LedGreen project and rename the *LedGreen.xxx* files to *TrafficLight.xxx*.

The lights can be switched on and off with the pushbutton switch from the previous examples. The timing for the change from red to green and green to red is managed with a [Timer](#) TimerGreenRed and the duration of the yellow light is managed with a second Timer TimerYellow.

The project TrafficLight.b4r is available in the SourceCode folder.

5.1.1 Sketch



Material:

- 1 pushbutton switch
- 1 green Led
- 1 yellow Led
- 1 red led
- 3 220 Ω resistors

We

- add a yellow and a red Led similar to the green one.
- connect the yellow led (+) to digital pin **8**.
- connect the red led (+) to digital pin **9**.

5.1.2 Code

```

Sub Process_Globals
  Public Serial1 As Serial

  Private pinButton As Pin           'pin for the button
  Private pinLEDGreen, pinLEDYellow, pinLEDRed As Pin  'pins for the Leds

  Private TimerGreenRed, TimerYellow As Timer
  Private LightOn = False As Boolean
  Private LightGreen = False As Boolean
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub

```

We declare the button and the three led Pins, the two Timers and two global variables LightOn and LightGreen.

```

LightOn = False    > light OFF
LightGreen = True  > green light ON

```

```

Private Sub AppStart
  Serial1.Initialize(115200)

  TimerGreenRed.Initialize("TimerGreenRed_Tick", 2000)
  TimerYellow.Initialize("TimerYellow_Tick", 500)

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
  pinLEDYellow.Initialize(8, pinLEDYellow.MODE_OUTPUT)
  pinLEDRed.Initialize(9, pinLEDRed.MODE_OUTPUT)
End Sub

```

We initialize TimerGreenRed with the Tick event ("TimerGreenRed_Tick" and an interval of 2000 which means that the Tick event is raised every 2 seconds (2000 milli seconds).

We initialize TimerYellow with the Tick event ("TimerYellow_Tick" and an interval of 500 which means that the Tick event is raised every 0.5 second (500 milli seconds).

We keep the code for the button and the green Led, initialize pinLEDYellow as digital pin **8** as output and initialize pinLEDRed as digital pin **9** as output.

The code is hopefully self-explanatory.

```

Private Sub pinButton_StateChanged (State As Boolean)
' Log("State: ", State)           'Log the State value

  If State = False Then           'if State = False
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      pinLEDRed.DigitalWrite(True) 'switch ON the red LED
      LightOn = Not(LightOn)       'change the value of LightOn
      BounceTime = Millis
'   Log("Light: ", LightOn)       'Log the LightOn value

      TimerGreenRed.Enabled = LightOn 'enable TimerGreenRed Timer

      If LightOn = False Then       'if LightOn = False
        pinLEDGreen.DigitalWrite(False) 'switch OFF LED Green
        pinLEDYellow.DigitalWrite(False) 'switch OFF LED Yellow
        pinLEDRed.DigitalWrite(False) 'switch OFF LED Red
      End If
    End If
  End If
End Sub

Private Sub TimerGreenRed_Tick
  If LightGreen = True Then        'if LightGreen = True
'   Log("TimerGreenRed_Tick LightYellow") 'write the Log
    TimerYellow.Enabled = True     'enable TimerYellow
    pinLEDGreen.DigitalWrite(False) 'switch OFF LED Green
    pinLEDYellow.DigitalWrite(True) 'switch ON LED Yellow
    LightGreen = False             'set LightGreen to False
  Else
'   Log("TimerGreenRed_Tick LightGreen") 'write the Log
    pinLEDRed.DigitalWrite(False)  'switch OFF LED Red
    pinLEDGreen.DigitalWrite(True)  'switch ON LED Green
    LightGreen = True              'set LightGreen to True
  End If
End Sub

Private Sub TimerYellow_Tick
' Log("TimerYellow_Tick LightRed") 'write the Log
' Log(" ")
  pinLEDYellow.DigitalWrite(False) 'switch OFF LED Yellow
  pinLEDRed.DigitalWrite(True)     'switch ON LED Red
  TimerYellow.Enabled = False      'disable Timer TimerYellow
End Sub

```

We can switch ON or OFF the traffic lights with the pushbutton switch.

Instead of a Timer we could have used CallSubPlus, it's equivalent to a Timer but fired only one time. CallSubPlus is used in the [PulsWidthModulation](#) project.

5.2 LightDimmer.b4r

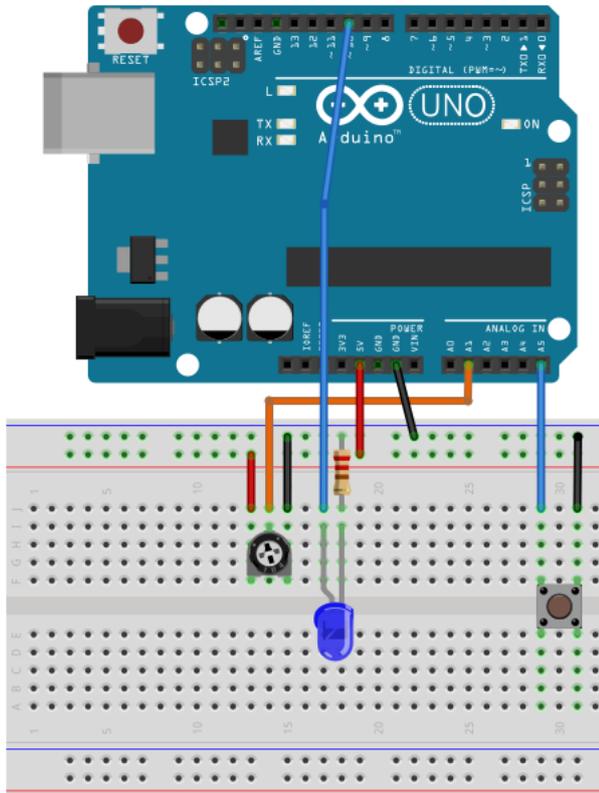
In this project we use a button, a led and a potentiometer.

With the pushbutton we switch the light ON or OFF.

With the potentiometer we modulate the brightness of a LED.

The project LightDimmer.b4r is available in the SourceCode folder.

5.2.1 Sketch



Material:

- 1 pushbutton switch
- 1 blue LED
- 1 potentiometer
- 1 220 Ω resistor

- The pushbutton switch is the same as in the previous examples.
One pin on **GRD** and the other one on pin **A5**.

- The blue LED is connected similar like in the other examples.

Cathode (-) via a 220 Ω resistor to **GND**.
Anode (+) to digital pin **~10**.

This pin allows [PWM](#) analog outputs.

We connect the **5V** pin to the + line of the breadboard.

- add a potentiometer which acts as a voltage divider.

- connect one of its ends to the **5V** line.

- connect the other end to the **GND** line.

- connect the slider to pin **A1**.

5.2.2 Code

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton, pinPot, pinLedBlue As Pin
  Private Reading = False As Boolean
  Private Timer1 As Timer
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub
```

We declare the three Pins, a global variable Reading which is True when we are reading and a Timer to read every 200 ms the value of pin **A1** and write its value to digital pin **~10** to dim or brighten the light.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLedBlue.Initialize(10, pinLedBlue.MODE_OUTPUT)

  pinPot.Initialize(pinPot.A1, pinPot.MODE_INPUT)

  Timer1.Initialize("Timer1_Tick", 200)
End Sub
```

We initialize the three pins and the Timer.

```
Sub pinButton_StateChanged (State As Boolean)
  Log("state: ", State)
  'state will be False when the button is clicked because of the PULLUP mode.
  If State = False Then
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      Reading = Not(Reading)
      BounceTime = Millis
      Timer1.Enabled = Reading
      Log("Reading: ", Reading)
      If Reading = False Then
        pinLedBlue.AnalogWrite(0)
      End If
    End If
  End If
End Sub
```

When State = **False** (button down) we invert the value of Reading,
Set Timer Timer1.Enabled = Reading.
And if Reading = **False** (stop reading) switch off the light pinLedBlue.AnalogWrite(0).

```
Private Sub Timer1_Tick
    Private Value As UInt
    Value = pinPot.AnalogRead
    Log("Value = ", Value)
    pinLedBlue.AnalogWrite(Value / 4)
End Sub
```

We declare a local variable `Value` and read the value of the pot slider on pin **A1**.

Then we write `Value / 4` to `pinLedBlue`.

We write `Value` to the Logs.

We need to divide `Value` by 4 because the max value of an analog input is 1023 (10 bits) and the max value we can write to an analog output is 255 (8 bits).

We could also use this shorter code.

```
Private Sub Timer1_Tick
    pinLedBlue.AnalogWrite(pinPot.AnalogRead / 4)
End Sub
```

5.3 DCMotor.b4r

In this project we use a pushbutton switch, a DC motor, a potentiometer and a 9V battery. With the pushbutton we switch the motor on or off. With the potentiometer we modulate the speed of the motor.

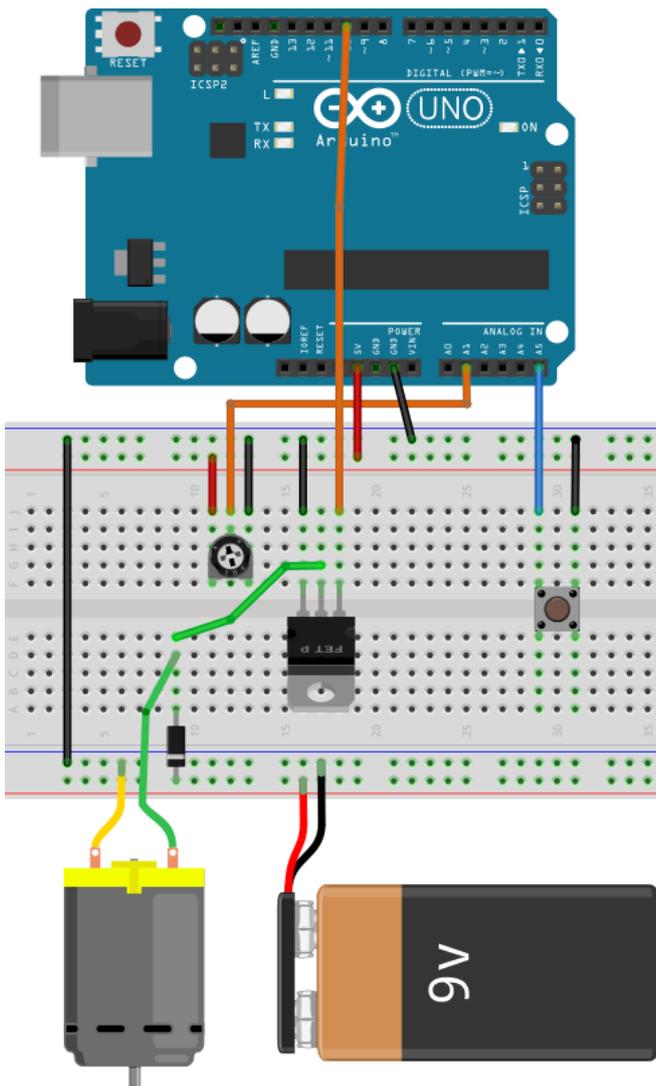
The project is based on the LightDimmer project, the DC motor ([TFK-280SA-22125](#)) replaces the blue Led and we need some more components to drive the motor.

- A 9V battery providing the power. The motor needs more power than the Arduino can provide.
- A MOSFET power transistor ([IRF520NPbF](#)), which acts as a 'power switch'.
- A diode ([1N4007](#)) to protect the motor.

It is inspired by a project in the Arduino Starter Kit.

The project DCMotor.b4r is available in the SourceCode folder.

5.3.1 Sketch



Material:

- 1 push button switch
- 1 DC motor [TFK-280SA-22125](#)
- 1 9V battery
- 1 potentiometer
- 1 MOSFET transistor [IRF520NPbF](#)
- 1 diode [1N4007](#)

The button and the potentiometer are the same as in the LightDimmer project.

First we connect the two GND lines of the breadboard. The GND of the Arduino board and the GND (-) of the 9V battery must be at the same level.

Then we add the MOSFET transistor, the motor, the diode, the 9V battery and the connecting wires like in the diagram.

We use the digital output pin ~10 to drive the motor.

5.3.2 Code

And the code is almost the same as the LightDimmer project.

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton, pinPot, pinMotor As Pin
  Private Reading = False As Boolean
  Private Timer1 As Timer
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub
```

We declare the three Pins, a global variable Reading which is True when we are reading and a Timer to read every 200 ms the value of pin **A1**.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinMotor.Initialize(10, pinMotor.MODE_OUTPUT)

  Timer1.Initialize("Timer1_Tick", 200)

  pinPot.Initialize(pinPot.A1, pinPot.MODE_INPUT)
End Sub
```

We initialize the three pins and the Timer.

```
Sub pinButton_StateChanged (State As Boolean)
  Log("state: ", State)
  'state will be False when the button is clicked because of the PULLUP mode.
  If State = False Then
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      Reading = Not(Reading)
      BounceTime = Millis
      Timer1.Enabled = Reading
      Log("Reading: ", Reading)
      If Reading = False Then
        pinMotor.AnalogWrite(0)
      End If
    End If
  End If
End Sub
```

When State = **False** (button down) we invert the value of Reading, Set the Timer Timer1.Enabled = Reading. And if Reading = **False** (stop reading) switch off the motor.

```
Private Sub Timer1_Tick
    Private Value As UInt
    Value = pinPot.AnalogRead
    Log("Value = ", Value)
    pinMotor.AnalogWrite(Value / 4)
End Sub
```

We declare a local variable `Value` and read the value of the pot slider on pin **A1**.

Then we write `Value / 4` to `pinMotor`.

We need to divide `Value` by 4 because the max value of an analog input is 1023 (10 bits) and the max value we can write to an analog output is 255 (8 bits).

5.4 DCMotorHBridge.b4r

This example is also inspired by an Arduino Starter Kit example.

In the previous example we could modulate the speed only in one direction. In the example below we can change the direction of the motor with a pushbutton switch.

With the right pushbutton we switch the motor ON or OFF.

With the left pushbutton we change the motor direction.

With the potentiometer we modulate the motor speed.

The project DCMotorHBridge.b4r is available in the SourceCode folder.

We use a specialized [H-bridge](#) integrated circuit to supply the power to the DC motor, a [L293D](#) circuit. This circuit allows to change the motion direction of the motor.

This circuit needs two power supplies, one 5V supply for its internal logic circuits, pin 16, and a power supply for the motor (5 to 36V), a 9V battery in our example (pin 8).

Pins 4 and 5 are the GND pins, the two grounds (5V and 9V must be at the same level).

Pin 2 is used to manage the motor speed, a PWM signal.

Pin 1 and pin 7 are used to change the motion direction.

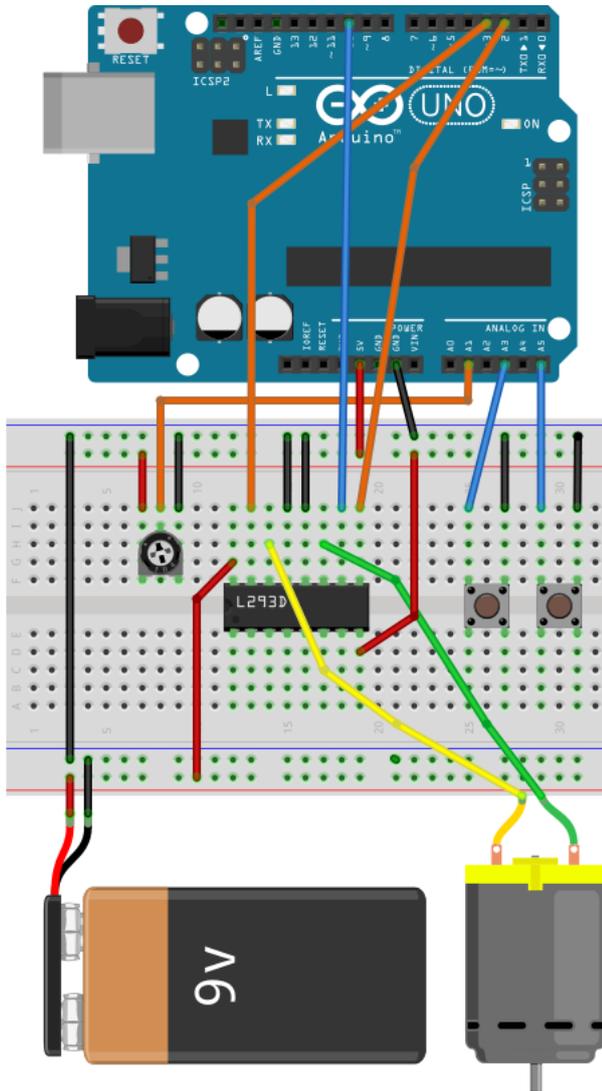
Pin 1 LOW and pin 7 HIGH one direction.

Pin 1 HIGH and pin 7 LOW the opposite direction.

Both pins LOW or HIGH, the motor stops.

Pin 3 and pin 6 provide the power for the motor.

5.4.1 Sketch



Material:

- 2 push button switches
- 1 DC motor [TFK-280SA-22125](#)
- 1 9V battery
- 1 potentiometer
- 1 IC [L293D](#) circuit

We connect:

Arduino **GND** pin to **GND** line of the breadboard.

Arduino **5V** pin to **5V** line of the breadboard.

The **GND** wire of the 9V battery to other **GND** line of the breadboard.

The **9V** wire of the 9V battery to **9V** line of the breadboard.

Both **GND** lines of the breadboard.

Both switches, one side to the **GND** line.

The other pin of the right switch, to switch ON or OFF the motor, to analog pin **A5**.

The other pin of the left switch, to change the direction of the motor, to analog pin **A3**.

The potentiometer, one end to the **GND** line, the other end to the **5V** line and the slider to analog pin **A1**.

For the IC (pin 1 is the upper right one):

Pin 1 to digital **2**.

Pin 2 to digital **~10**.

Pin 3 to one wire of the DC motor.

Pin 4 and 5 to the **GND** line.

Pin 6 to the second wire of the DC motor.

Pin 7 to digital pin **3**.

Pin 16 to the **9V** line.

5.4.2 Code

Sub Process_Globals

```
Public Serial1 As Serial

Private MotorON = False As Boolean      'motor ON or OFF, ON = True
Private MotorDirection = False As Boolean 'motor direction
Private MotorSpeed = 0 As UInt

Private TimerMotor As Timer
Private pinSwitchMotorONOFF, pinSwitchMotorDirection, pinMotorSpeed,
pinMotorControl1, pinMotorControl2, pinPot As Pin

Private BounceTimeMotorON, BounceTimeMotorDirection As ULong
Private BounceDelay = 10 As ULong
End Sub
```

We declare different variables and the different Pins:

Variables:

- MotorON true motor ON, False motor OFF.
- MotorDirection True one direction, False the other direction.
- MotorSpeed value for the motor speed.

Pins:

- pinSwitchMotorONOFF to the switch the motor ON or OFF
- pinSwitchMotorDirection to change the motor motion direction.
- pinMotorSpeed to change the motor speed.
- pinMotorControl1 and pinMotorControl2 to change the motor motion direction.
- pinPot for the potentiometer slider.

We use a Timer, TimerMotor, to read the value of the position of the potentiometer slider, calculate the motor speed value and write it to the motor speed pin.

We use two variables BounceTimeMotorON and BounceTimeMotorDirection for the anti-bouncing of the switches.

```

Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  TimerMotor.Initialize("TimerMotor_Tick", 200)

  pinSwitchMotorONOFF.Initialize(pinSwitchMotorONOFF.A5,
pinSwitchMotorONOFF.MODE_INPUT_PULLUP)
  pinSwitchMotorONOFF.AddListener("pinSwitchMotorONOFF_StateChanged")
  pinSwitchMotorDirection.Initialize(pinSwitchMotorONOFF.A3,
pinSwitchMotorDirection.MODE_INPUT_PULLUP)
  pinSwitchMotorDirection.AddListener("pinSwitchMotorDirection_StateChanged")
  pinMotorControl1.Initialize(2, pinMotorControl1.MODE_OUTPUT)
  pinMotorControl2.Initialize(3, pinMotorControl1.MODE_OUTPUT)
  pinMotorSpeed.Initialize(9, pinMotorSpeed.MODE_OUTPUT)

  pinPot.Initialize(pinPot.A1, pinPot.MODE_INPUT)

  pinMotorControl1.DigitalWrite(MotorDirection)
  pinMotorControl2.DigitalWrite(Not(MotorDirection))

```

End Sub

We initialize the different pins and add the event routine declarations.

```

Private Sub pinSwitchMotorONOFF_StateChanged (State As Boolean)
  If State = False Then
    If Millis - BounceTimeMotorON < BounceDelay Then
      Return
    Else
      MotorON = Not(MotorON)
      If MotorON = True Then
        TimerMotor.Enabled = MotorON
      Else
        pinMotorSpeed.AnalogWrite(0)
      End If
      BounceTimeMotorON = Millis
    End If
    Log("MotorON: ", MotorON)
  End If
End Sub

```

Similar to the **StateChange** event routines in the other projects, the same for **Private Sub pinSwitchMotorDirection_StateChanged (State As Boolean)**

```

Private Sub TimerMotor_Tick
  If MotorON = True Then
    MotorSpeed = pinPot.AnalogRead / 4
    Log("MotorSpeed: ", MotorSpeed)
    pinMotorSpeed.AnalogWrite(MotorSpeed)
  End If
End Sub

```

Similar to the **Timer** event routines in the other projects.

5.5 ServoMotor.b4r

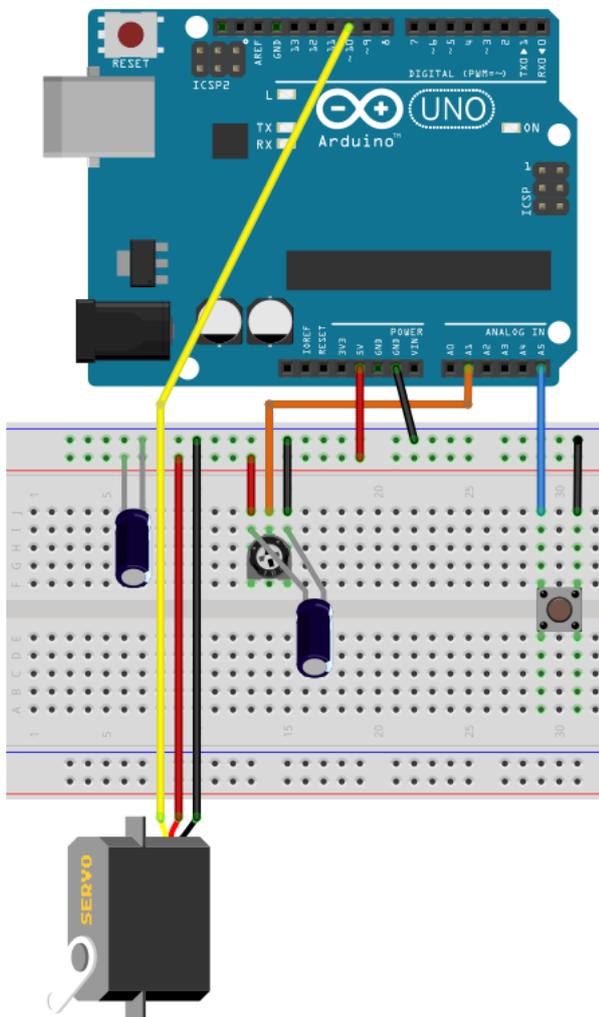
In this project we use a pushbutton switch, a servo motor and a potentiometer. With the pushbutton we switch the motor on or off. With the potentiometer we modulate the position of the servo motor. The servo motor position range is between 0 and 180°.

The project is based on the LightDimmer project, the servo motor Spring RC SM - S2309S replaces the blue Led.

It is inspired by a project in the Arduino Starter Kit.

The project ServoMotor.b4r is available in the SourceCode folder.

5.5.1 Sketch



Material:

- 1 pushbutton switch
- 1 potentiometer
- 1 servo motor
- 2 100 μ F capacitors

The pushbutton switch is the same as in the previous examples.

One pin on **GRD** and the other one on pin **A5**.

The potentiometer has:

One end connected to the **5V** line, the other end to the **GND** line and the slider to pin **A1**.

We add a servo motor.

- connect the red wire to the **5V** line.
- connect the black wire to the **GND** line.
- connect the white wire to digital pin **~10**.

This pin allows [PWM](#) analog outputs.

5.5.2 Code

Sub Process_Globals

```
Public Serial1 As Serial
Private pinButton, pinPot, pinMotor As Pin
Private Reading = False As Boolean
Private Timer1 As Timer
Private Angle = 0 As UInt
Private BounceTime As ULong
Private BounceDelay = 10 As ULong
End Sub
```

Private Sub AppStart

```
Serial1.Initialize(115200)
Log("AppStart")

'Using the internal pull up resistor to prevent the pin from floating.
pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
pinButton.AddListener("pinButton_StateChanged")

pinMotor.Initialize(10, pinMotor.MODE_OUTPUT)

Timer1.Initialize("Timer1_Tick", 200)

pinPot.Initialize(pinPot.A1, pinPot.MODE_INPUT)
End Sub
```

Similar to the other projects.

Sub pinButton_StateChanged (State As Boolean)

```
Log("state: ", State)
'state will be False when the button is clicked because of the PULLUP mode.
If State = False Then
    If Millis - BounceTime < BounceDelay Then
        Return
    Else
        Reading = Not(Reading)
        BounceTime = Millis
        Timer1.Enabled = Reading
        Log("Reading: ", Reading)
        pinMotor.AnalogWrite(128)
    End If
End If
End Sub
```

Similar as in the other projects.

```
Private Sub Timer1_Tick
    Private Value As UInt
    Value = pinPot.AnalogRead
    ' Angle = MapRange(Value, 0, 1023, 0, 180)
    ' Angle = MapRange(Value, 0, 1023, 0, 255)
    Angle = MapRange(Value, 0, 1023, 40, 225)
    Log("Angle = ", Angle)
    pinMotor.AnalogWrite(Angle)
End Sub
```

We use a timer to read the output of the pot slider at analog pin **A1**.

Then we calculate an angle and write it to digital pin **10**.

In the Arduino Starter Kit documentation, the angle written should be between 0 and 180.

Unfortunately, the position range is smaller than 180° and with values near 0 the behavior is not OK.

I haven't found any data sheet specifying the input signal.

I tried with values between 0 and 255 but with values near 0 and with values near 255 the behavior is not correct.

My experience to have a correct behavior are values between 40 and 225 for a position range of 180°.

5.6 PulseWidthModulation.b4r

This project shows the principle of PulseWidthModulation.

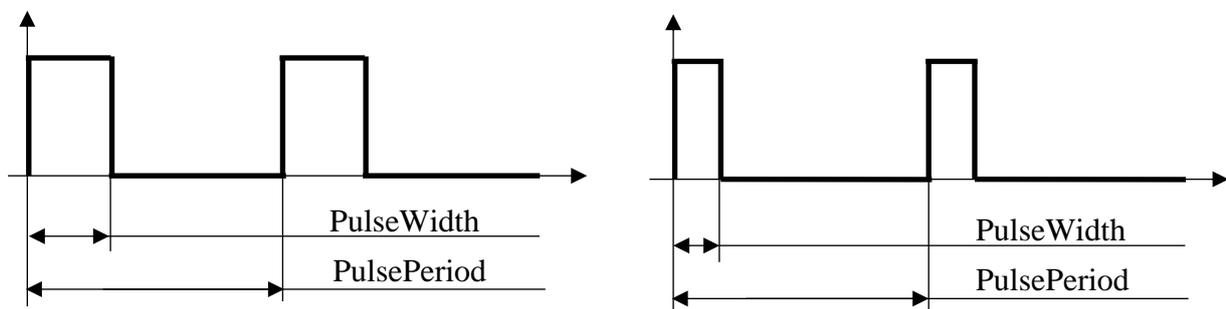
The pulse period is constant, and the pulse width can have a value between 0 (0% modulation) and the pulse period length (100% modulation).

The value of PulsePeriod can be set in the program and the PulseWidth is calculated in function of a potentiometer slider position.

It is based on the LightDimmer project.

The project PulseWidthModulation.b4r is available in the SourceCode folder.

Pulse Width Modulation:



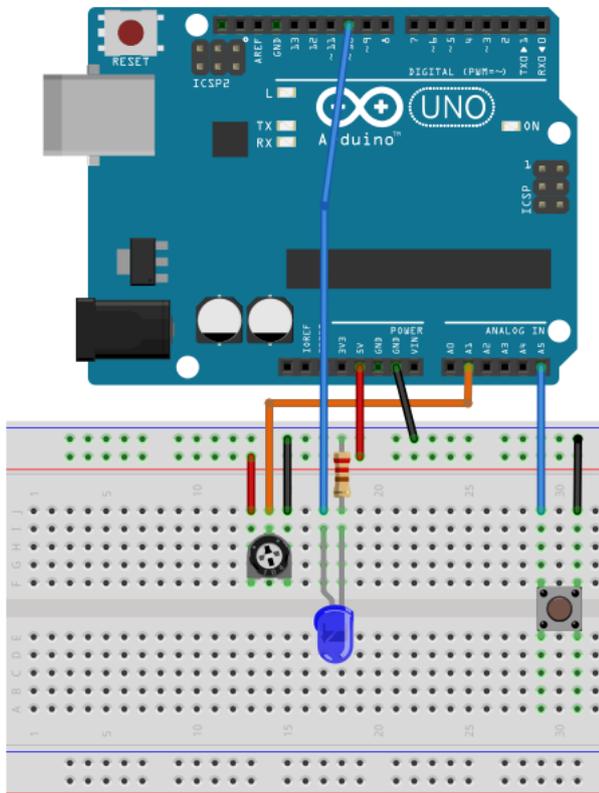
PulsePeriod can be defined in the code and remains constant.

PulseWidth is variable, between 0 (0%) and PulsePeriod (100% modulation).

I used a LED in the project to show the principle with relatively big value for PulsePeriod.

The principle is the same as the PWM output of the digital pins with the “~” prefix, but the pulse period is much smaller.

5.6.1 Sketch



The circuit is the same as the LightDimmer circuit.

Material:

- 1 push button switch
- 1 LED
- 1 potentiometer
- 1 220 Ω resistor

Pushbutton, one pin connected to the **GND** line of the breadboard and the other pin connected to analog pin **5** of the Arduino ONE.

LED cathode (-) connected to the **GND** line of the breadboard via a 220 Ω resistor.

LED anode (+) connected to digital pin **10**.

Potentiometer, one pin connected to the **GND** line of the breadboard, the other pin connected to the **5V** line of the breadboard, and the slider connected to analog pin **2** of the Arduino ONE.

5.6.2 Code

We use a Timer `TimerPulsePeriod` to generate each pulse with the period `PulsePeriod`. In the `TimerPulsePeriod_Tick` routine we switch the LED ON and call `CallSubPlus` to switch the LED OFF after the time `PulseWidth` has elapsed.

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton, pinPot, pinLED As Pin
  Private StateON_OFF = False As Boolean 'reading ON or OFF
  Private TimerPulsePeriod As Timer      'Timer for PulsePeriod
  Private PulseWidth As ULong            'calculated in TimerPulse_Tick
  Private PulsePeriod = 500 As ULong     'remains constant
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub
```

We define the different objects and variables.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLED.Initialize(10, pinLED.MODE_OUTPUT)

  pinPot.Initialize(pinPot.A2, pinPot.MODE_INPUT)

  TimerPulsePeriod.Initialize("TimerPulsePeriod_Tick", PulsePeriod)
End Sub
```

We initialize the different objects.

```
Private Sub pinButton_StateChanged (State As Boolean)
  ' Log("state: ", State)
  'State will be False when the button is clicked because of the PULLUP mode.
  If State = False Then
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      StateON_OFF = Not(StateON_OFF)
      BounceTime = Millis
      TimerPulsePeriod.Enabled = StateON_OFF
    '   Log("Reading: ", Reading)      'Log for testing
      If StateON_OFF = False Then
        pinLED.DigitalWrite(False)
      End If
    End If
  End If
End Sub
```

With the pushbutton switch we enable or disable `TimerPulsePeriod` to switch the system ON or OFF.

```

Private Sub TimerPulsePeriod_Tick
' Log(pinPot.AnalogRead)           'Log for testing

'calculate PulseWidth in function of the pot slider
'Map the min max values:
'Min:  0 pot > PulseWidth = 0
'Max: 1023 pot > PulseWidth = PulsePeriod
PulseWidth = MapRange(pinPot.AnalogRead, 0, 1023, 0, PulsePeriod)
If PulseWidth = PulsePeriod Then '100% modulation
    pinLED.DigitalWrite(True)     'switch the LED ON
Else If PulseWidth = 0 Then      '0% modulation
    pinLED.DigitalWrite(False)    'switch the LED OFF
Else
    pinLED.DigitalWrite(True)     'switch the LED ON
    CallSubPlus("EndPulse", PulseWidth, 0)
End If
End Sub

```

We:

Calculate PulseWidth.

pinPot.AnalogRead can have a value between 0 and 1023.

PulseWidth can have values between 0 and PulsePeriod.

We use `MapRange(pinPot.AnalogRead, 0, 1023, 0, PulsePeriod)` to map the value of the potentiometer slider to the correct value of PulseWidth.

Switch the LED ON or OFF depending on the value of PulseWidth.

Call EndPulse to switch the LED OFF after the PulseWidth time has elapsed.

```

Private Sub EndPulse(Tag As Byte)
    pinLED.DigitalWrite(False)     'switch the LED OFF
' Log("EndPulse")                 'Log for testing
End Sub

```

We switch the LED OFF, pulse end.

5.7 PulsePeriodModulation.b4r

This project shows the principle of PulsePeriodModulation. I use this name; I didn't find any other.

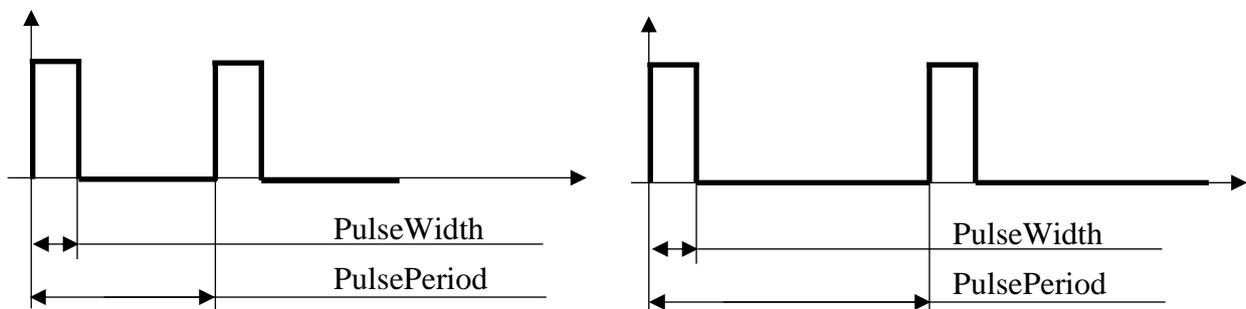
The difference between PulsePeriodModulation and PulseWidthModulation is that in PulsePeriodModulation the pulse width is constant, and the pulse period is variable and can have a value between pulse width and a max. pulse period value.

The value of PulseWidth can be set in the program and PulsePeriod is calculated in function of the potentiometer slider position.

It is based on the LightDimmer project.

The project PulsePeriodModulation.b4r is available in the SourceCode folder.

PulsePeriodModulation:



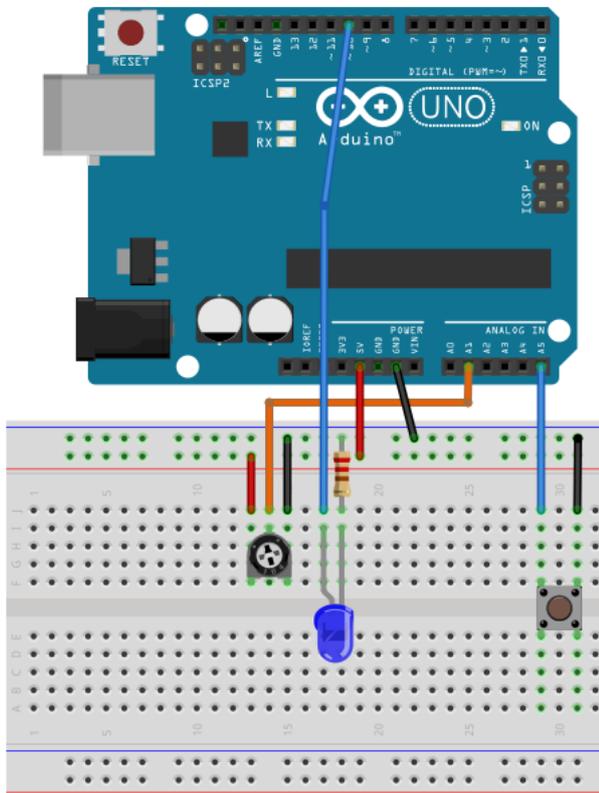
PulseWidth can be defined in the code and remains constant.

PulsePeriod is variable, between PulseWidth and a max. value.

This kind of modulation can be used for DC motors with very slow motion.

I used a LED in the project to show the principle with big values for PulseWidth and PulsePeriod. For a motor, these values must be much smaller.

5.7.1 Sketch



The circuit is the same as the LightDimmer circuit.

Material:

- 1 push button switch
- 1 LED
- 1 potentiometer
- 1 220 Ω resistor

Pushbutton, one pin connected to the **GND** line of the breadboard, and the other pin connected to analog pin **5** of the Arduino ONE.

LED cathode (-) connected to the **GND** line of the breadboard via a 220 Ω resistor.

LED anode (+) connected to digital pin **10**.

Potentiometer, one pin connected to the **GND** line of the breadboard, the other pin connected to the **5V** line of the breadboard, and the slider connected to analog pin **2** of the Arduino ONE.

5.7.2 Code

We define the different objects and variables.

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton, pinPot, pinLED As Pin
  Private StateON_OFF = False As Boolean      'state ON or OFF
  Private MaxPulsePeriod = 2000 As ULong     'Max PulsePeriod value
  Private PulseWidth = 200 As ULong          'remains constant
  Private PulsePeriod As ULong                'calculated in TimerPulse_Tick
  Private BounceTime As ULong
  Private BounceDelay = 10 As ULong
End Sub
```

Program start:

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLED.Initialize(10, pinLED.MODE_OUTPUT)

  pinPot.Initialize(pinPot.A2, pinPot.MODE_INPUT)
End Sub
```

pinButton Routine:

```
Private Sub pinButton_StateChanged (State As Boolean)
  ' Log("state: ", State)
  'state will be False when the button is clicked because of the PULLUP mode.
  If State = False Then
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      StateON_OFF = Not(StateON_OFF)
      Log("StateON_OFF: ", StateON_OFF)
      BounceTime = Millis
      If StateON_OFF = True Then
        NextPulse(0)
      Else
        pinLED.DigitalWrite(False)
      End If
    End If
  End If
End Sub
```

We use two routines to switch the LED ON or OFF called with CallSubPlus.

With CallSubPlus, the given routine is called after the given time.

CallSubPlus (SubName , DelayMs, Tag)

Tag is the parameter in the routine.

NextPulse routine:

```
Private Sub NextPulse(Tag As Byte)
  If StateON_OFF = True Then
    PulsePeriod = MapRange(pinPot.AnalogRead, 0, 1023, PulseWidth, MaxPulsePeriod)
    If PulsePeriod >= MaxPulsePeriod - 1 Then
      pinLED.DigitalWrite(False)
    Else if PulsePeriod <= PulsePeriod + 1 Then
      pinLED.DigitalWrite(True)
    Else
      pinLED.DigitalWrite(True)
      CallSubPlus("EndPulse", PulseWidth, 0)
    End If
    CallSubPlus("EndPulse", PulseWidth, 0)
    CallSubPlus("NextPulse", PulsePeriod, 0)
  '   Log("NextPulse")                               'Log for testing
  End If
End Sub
```

We:

Calculate PulsePeriod.

pinPot.AnalogRead can have a value between 0 and 1023.

PulsePeriod can have values between PulseWidth and MaxPulsePeriod.

We use MapRange(pinPot.AnalogRead, 0, 1023, PulseWidth, MaxPulsePeriod) to map the value of the potentiometer slider to the correct value of PulsePeriod.

Switch the LED ON or OFF depending on the value of PulsePeriod.

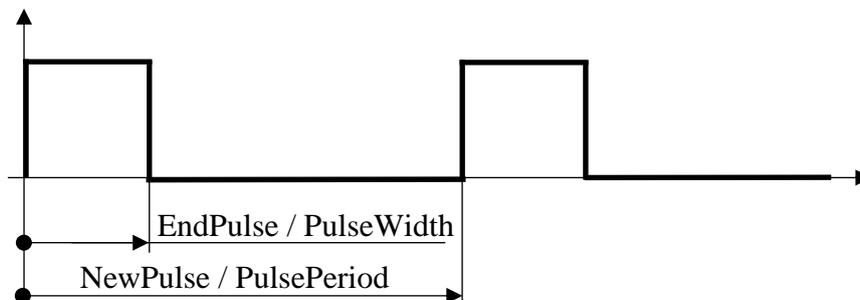
Call EndPulse to switch the LED OFF after the PulseWidth time has elapsed.

Call NextPulse to generate the next pulse.

EndPulse routine:

```
Private Sub EndPulse(Tag As Byte)
  pinLED.DigitalWrite(False) 'switch the LED OFF
  ' Log("EndPulse")         'Log for testing
End Sub
```

We switch the LED OFF, pulse end.



5.8 DCMotor slow motion

This project is a combination of the DCMotorHBRidge and the PulsePeriodModulation projects.

With the right pushbutton we switch the motor on or off.

With the left pushbutton we change the motion direction.

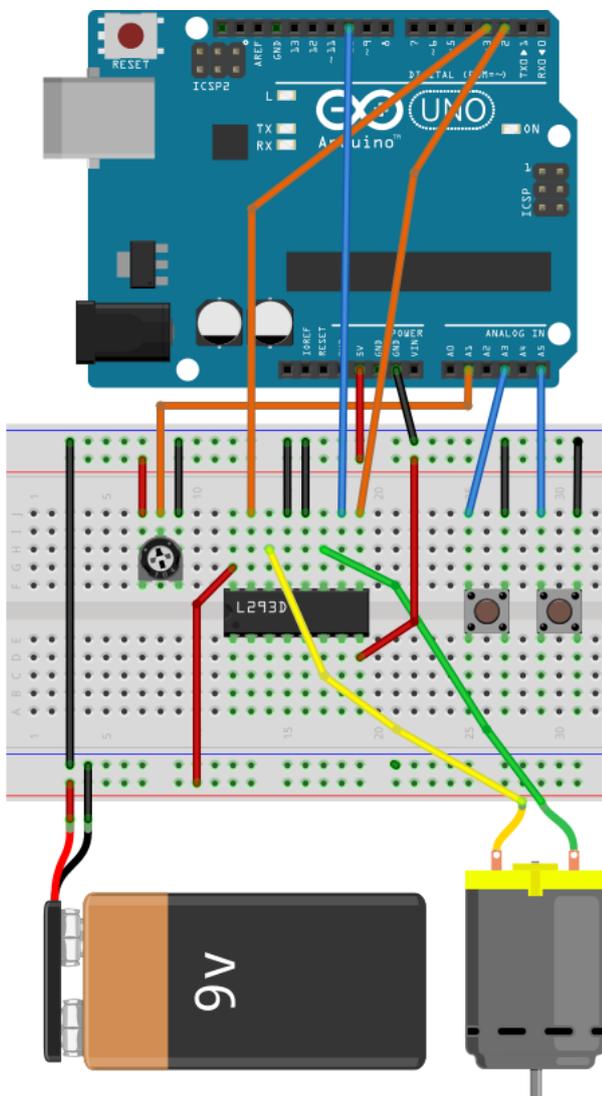
With the potentiometer we modulate the speed of the motor with very slow motion.

I had used this kind of modulation, quite some years ago, for the power supply of a railroad model.

The project DCMotorSlowMotion.b4r is available in the SourceCode folder.

5.8.1 Sketch

The circuit is exactly the same as the DCMotorHBRidge project.



Material:

- 2 push button switches
- 1 DC motor [TFK-280SA-22125](#)
- 1 9V battery
- 1 potentiometer
- 1 [L293D](#) IC circuit

We connect:

Arduino **GND** pin to **GND** line of the breadboard.

Arduino **5V** pin to **5V** line of the breadboard.

The **GND** wire of the 9V battery to other **GND** line of the breadboard.

The **9V** wire of the 9V battery to **9V** line of the breadboard.

Both **GND** lines of the breadboard.

Both switches, one side to the **GND** line.

The other pin of the right switch, to switch ON or OFF the motor, to analog pin **A5**.

The other pin of the left switch, to change the direction of the motor, to analog pin **A3**.

The potentiometer, one end to the **GND** line, the other end to the **5V** line and the slider to analog pin **A1**.

For the IC (pin 1 is the upper right one):

Pin **1** to digital **2**.

Pin **2** to digital **~10**.

Pin **3** to one wire of the DC motor.

Pin **4** and **5** to the **GND** line.

Pin **6** to the second wire of the DC motor.

Pin **7** to digital pin **3**.

Pin **16** to the **9V** line.

5.8.2 Code

The code is almost the same as in the DCMotorHBridge project.

The difference is that we replace the timer by two routines from the PulsePeriodModulation project `NextPulse` and `EndPulse`.

```
Sub Process_Globals
    Public Serial1 As Serial

    Private MotorON = False As Boolean           'motor ON or OFF, ON = True
    Private MotorDirection = False As Boolean    'motor direction
    Private MotorSpeed = 0 As UInt

    Private pinSwitchMotorONOFF, pinSwitchMotorDirection, pinMotorSpeed,
    pinMotorControl1, pinMotorControl2, pinPot As Pin

    Private BounceTimeMotorON, BounceTimeMotorDirection As ULong
    Private BounceDelay = 10 As ULong

    Private MaxPulsePeriod = 500 As ULong       'Max PulsePeriod value
    Private PulseWidth = 20 As ULong           'remains constant
    Private PulsePeriod As ULong               'calculated in NextPulse

End Sub
```

We remove the Timer and add the three variables:

- `MaxPulsePeriod` the max pulse period, a value of 500 instead of 2000.
- `PulseWidth` pulse width, this value remains constant, a value of 20 instead of 200.
- `PulsePeriod` pulse period, this value is calculated in the routine according to the pot slider.

You can change the values of `MaxPulsePeriod` and `PulseWidth` to see what happens or to adapt them to another motor.

```
Private Sub AppStart
    Serial1.Initialize(115200)
    Log("AppStart")

    pinSwitchMotorONOFF.Initialize(pinSwitchMotorONOFF.A5,
    pinSwitchMotorONOFF.MODE_INPUT_PULLUP)
    pinSwitchMotorONOFF.AddListener("pinSwitchMotorONOFF_StateChanged")
    pinSwitchMotorDirection.Initialize(pinSwitchMotorONOFF.A3,
    pinSwitchMotorDirection.MODE_INPUT_PULLUP)
    pinSwitchMotorDirection.AddListener("pinSwitchMotorDirection_StateChanged")
    pinMotorControl1.Initialize(2, pinMotorControl1.MODE_OUTPUT)
    pinMotorControl2.Initialize(3, pinMotorControl1.MODE_OUTPUT)
    pinMotorSpeed.Initialize(10, pinMotorSpeed.MODE_OUTPUT)

    pinPot.Initialize(pinPot.A1, pinPot.MODE_INPUT)

    pinMotorControl1.DigitalWrite(MotorDirection)
    pinMotorControl2.DigitalWrite(Not(MotorDirection))

End Sub
```

Almost the same as in the DCMotorHBridge, removed the Timer initialization.

```
Private Sub pinSwitchMotorONOFF_StateChanged (State As Boolean)
  If State = False Then
    If Millis - BounceTimeMotorON < BounceDelay Then
      Return
    Else
      MotorON = Not(MotorON)
      If MotorON = True Then
        NextPulse(0)
      Else
        pinMotorSpeed.AnalogWrite(0)
      End If
      BounceTimeMotorON = Millis
    End If
    Log("MotorON: ", MotorON)
  End If
End Sub
```

Almost the same as in the DCMotorHBridge, the line `NextPulse(0)` replaces `TimerMotor.Enabled = MotorON`.

```
Private Sub pinSwitchMotorDirection_StateChanged (State As Boolean)
' Log("StateON ", State)
  If State = False Then
    If Millis - BounceTimeMotorDirection < BounceDelay Then
      Return
    Else
      MotorDirection = Not(MotorDirection)
      pinMotorControl1.DigitalWrite(MotorDirection)
      pinMotorControl2.DigitalWrite(Not(MotorDirection))
      BounceTimeMotorDirection = Millis
    End If
    Log("MotorDirection: ", MotorDirection)
  End If
End Sub
```

Exactly the same as in the DCMotorHBridge project.

```
Private Sub NextPulse(Tag As Byte)
  If SateON_OFF = True Then
    PulsePeriod = MapRange(pinPot.AnalogRead, 0, 1023, PulseWidth, MaxPulsePeriod)
    If PulsePeriod >= MaxPulsePeriod - 1 Then
      pinMotor.DigitalWrite(False)
    Else If PulsePeriod <= PulseWidth + 1 Then
      pinMotor.DigitalWrite(True)
    Else
      pinMotor.DigitalWrite(True)
      CallSubPlus("EndPulse", PulseWidth, 0)
    End If
    CallSubPlus("EndPulse", PulsePeriod, 0)
    CallSubPlus("NextPulse", PulsePeriod, 0)
  '   Log("NextPulse")           'Log for testing
  End If
End Sub

Private Sub EndPulse(Tag As Byte)
  pinMotor.DigitalWrite(False) 'switch the LED OFF
  '   Log("EndPulse")           'Log for testing
End Sub
```

These two routines are exactly the same as in the PulsePeriodModulation project.

5.9 LCDDisplay.b4r

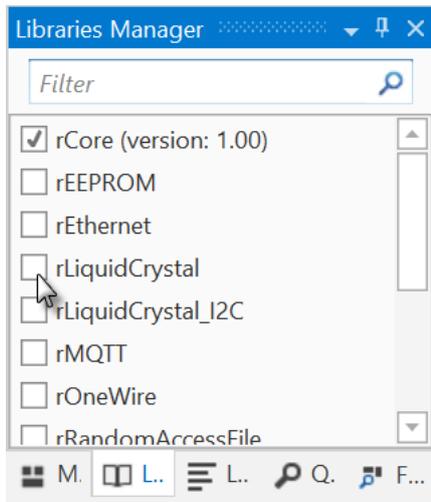
We reuse the TrafficLight project and add a LCD display to show the different states of the lights.

Make a new folder LCDDisplay, copy there all the files of the TrafficLight project and rename the *TrafficLight.xxx* files to *LCDDisplay.xxx*.

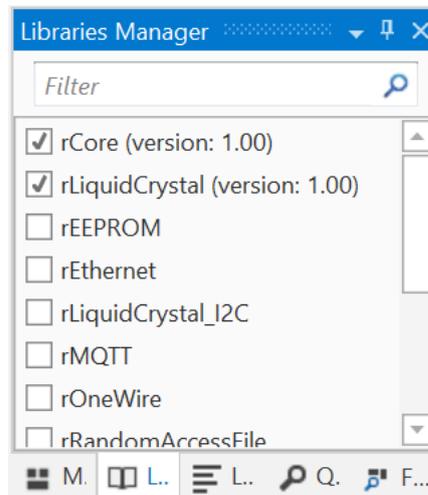
The project LCDDisplay.b4r is available in the SourceCode folder.

For this project we need the rLiquidCrystal library.

In the Libraries Tab select the rLiquidCrystal library.



Before



after.

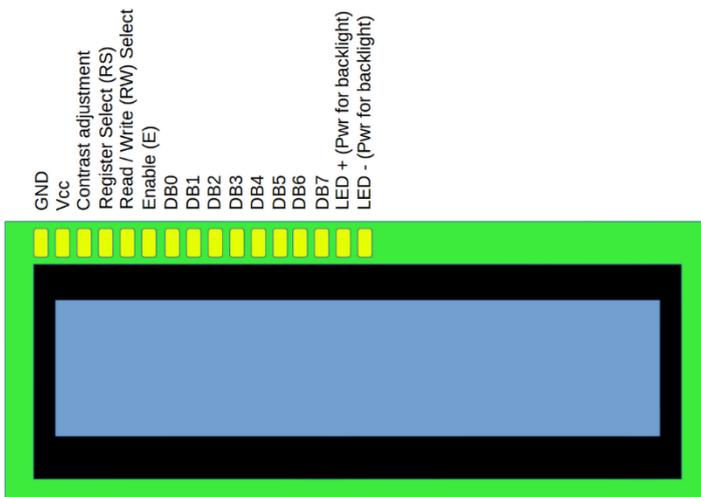
The library and its version number appear on top of the Library Manager.

The rLiquidCrystal library has following methods:

- **LCD.Initialize**(RS As Byte, RW As Byte, Enable As Byte, DataPins As Byte())
RS pin index of the Arduino for the display RS (Register Select) pin.
 Arduino pin D12 in our case.
RW pin index of the Arduino for the display RW (Read / Write) pin.
 Breadboard GDN line in our case.
Enable pin index of the Arduino for the display Enable pin.
 Arduino pin D12 in our case.
DataPins pins for the data.
 pins D5, D4, D3, D2 in our case.
- **LCD.Begin**(NumberOfColumns As Byte, NumberOfRows As Byte)
NumberOfColumns in our case 16
NumberOfRows in our case 2
- **LCD.Clear**
Clears the screen and sets the cursor to (0, 0). Top left corner.
- **LCD.SetCursor**(Column As Byte, Row As Byte)
Sets the cursor for display at the given Column and Row.
- **LCD.Write**(Message As Object)
Writes a message to the LCD display.
Message can be either a:
 - String
 - Number
 - Array of bytes.
- **LCD.CursorON**
LCD.CursorON = True sets an underline at the current cursor position.
- **LCD.DisplayON**
LCD.DisplayON = True sets the display state.
- **LCD.Blink**
LCD.Blink = True activates the blinking of the cursor.

5.9.1 LCD display

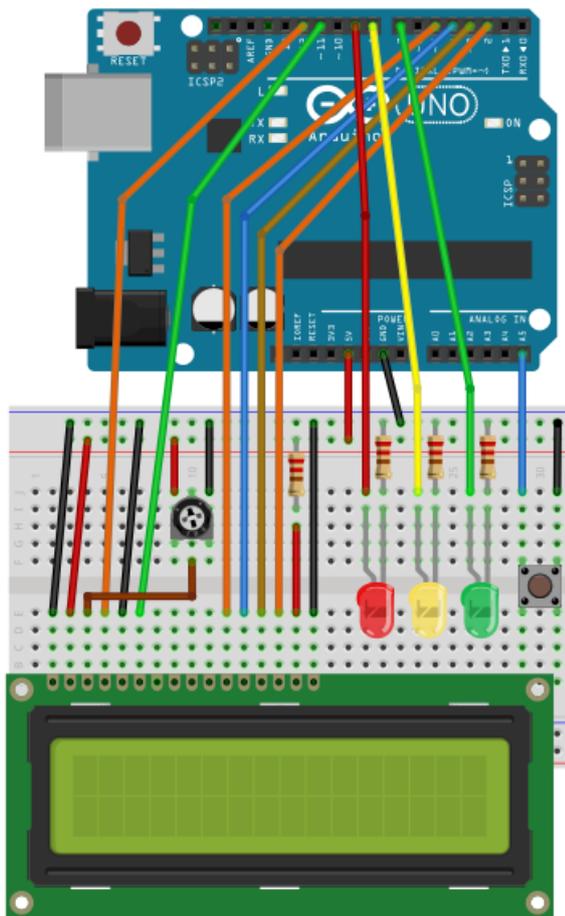
The LCD display has 2 rows of 16 characters.



The 16 pins: in the project connected to

GND	Power ground	breadboard GND line
Vcc	Power 5V	breadboard 5 V line
Vo	Contrast adjustment	slider of the potentiometer 0 min to 5V max.
RS	Register select	Arduino D12
RW	Read / Write	breadboard GND (Write)
E	Enable	Arduino D11
DB0	Digital pin 0	not used
DB1	Digital pin 1	not used
DB2	Digital pin 2	not used
DB3	Digital pin 3	not used
DB4	Digital pin 4	Arduino D5
DB5	Digital pin 5	Arduino D4
DB6	Digital pin 6	Arduino D3
DB7	Digital pin 7	Arduino D2
LED+	Backlight power +	breadboard 5 V line via a 220 Ω resistor
LED -	Backlight power -	breadboard GND line

5.9.2 Sketch



Material:

- 1 pushbutton switch
- 1 green Led
- 1 yellow Led
- 1 red led
- 1 potentiometer
- 1 LCD display
- 4 220 Ω resistors

Connect the different parts like in the sketch.

5.9.3 Code

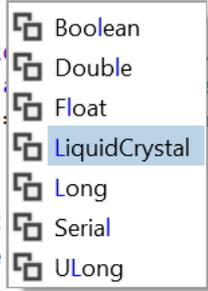
We need to add the definition of the liquid crystal display.

Type `Public lcdDisplay As`, the auto complete function suggests you the possible objects.

```

11 | Public pinButton As Pin      'pin for the button
12 | Public pinLEDGreen, pinLEDYellow, pinLEDRed As Pin
13 | Public lcdDisplay As L
14 |
15 | Public TimerGreenRed, TimerYellow As Timer
16 | Public LightOn = False As Boolean
17 | Public LightGreen = False As Boolean
18 | End Sub
19 |
20 | Private Sub AppStart
21 |     Serial1.Initialize
22 |

```



Click on `LiquidCrystal` to confirm. The rest of the code is hopefully self-explanatory.

Sub Process_Globals

```

Public Serial1 As Serial

Public pinButton As Pin      'pin for the button
Public pinLEDGreen, pinLEDYellow, pinLEDRed As Pin 'pins for the Leds
Public lcdDisplay As LiquidCrystal

Public TimerGreenRed, TimerYellow As Timer
Public LightOn = False As Boolean
Public LightGreen = False As Boolean
Private BounceTime As ULong
Private BounceDelay = 10 As ULong
End Sub

```

Private Sub AppStart

```

Serial1.Initialize(115200)

TimerGreenRed.Initialize("TimerGreenRed_Tick", 2000)
TimerYellow.Initialize("TimerYellow_Tick", 500)

'Using the internal pull up resistor to prevent the pin from floating.
pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
pinButton.AddListener("pinButton_StateChanged")

pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
pinLEDYellow.Initialize(8, pinLEDYellow.MODE_OUTPUT)
pinLEDRed.Initialize(9, pinLEDRed.MODE_OUTPUT)

'initialize the display(RS As Byte, RW As Byte, Enable As Byte
'RS pin > Arduino digital pin 12
'RW pin > 255 means not used
'Enable pin > Arduino digital pin 11
'DataPins: Arduino digital pins 5, 4, 3, 2
lcdDisplay.Initialize(12, 255, 11, Array As Byte(5, 4, 3, 2))
lcdDisplay.Begin(16, 2) 'set the display type 2 * 16 characters
lcdDisplay.Write("Light OFF") 'write "Light OFF" in the first line

```

End Sub

```

Private Sub pinButton_StateChanged (State As Boolean)
  Log("State: ", State)           'Log the Start value
  If State = False Then          'if State = False
    If Millis - BounceTime < BounceDelay Then
      Return
    Else
      pinLEDRed.DigitalWrite(True) 'switch ON the red LED
      LightOn = Not(LightOn)       'change the value of LightOn
      Log("Light: ", LightOn)     'Log the LightOn value
      BounceTime = Millis
      lcdDisplay.Clear             'clear the display
      If LightOn = True Then
        lcdDisplay.Write("Light ON") 'write "Light ON" in the first line
        lcdDisplay.SetCursor(0, 1)  'set the cursor at the begin of second line
        lcdDisplay.Write("Red")     'write "red" in the second line
      Else
        'if State = True
        lcdDisplay.Write("Light OFF") 'write "Light OFF" in the first line
      End If
      TimerGreenRed.Enabled = LightOn 'enable TimerGreenRed Timer

      If LightOn = False Then       'if LightOn = False
        pinLEDGreen.DigitalWrite(False) 'switch OFF LED Green
        pinLEDYellow.DigitalWrite(False) 'switch OFF LED Yellow
        pinLEDRed.DigitalWrite(False)  'switch OFF LED Red
      End If
    End If
  End If
End Sub

Private Sub TimerGreenRed_Tick
  If LightGreen = True Then        'if LightGreen = True
    Log("TimerGreenRed_Tick LightYellow") 'write the Log
    lcdDisplay.SetCursor(0, 1)     'position the cursor at the begin of second line
    lcdDisplay.Write("yellow")
    TimerYellow.Enabled = True
    pinLEDGreen.DigitalWrite(False) 'switch OFF LED Green
    pinLEDYellow.DigitalWrite(True) 'switch ON LED Yellow
    LightGreen = False             'set LightGreen to False
  Else
    'if LightGreen = False
    Log("TimerGreenRed_Tick LightGreen") 'write the Log
    lcdDisplay.SetCursor(0, 1)       'set the cursor at the begin of second line
    lcdDisplay.Write("green ")      'write "green " in the second line
    pinLEDRed.DigitalWrite(False)   'switch OFF LED Red
    pinLEDGreen.DigitalWrite(True)   'switch ON LED Green
    LightGreen = True               'set LightGreen to True
  End If
End Sub

Private Sub TimerYellow_Tick
  Log("TimerYellow_Tick LightRed") 'write the Log
  Log(" ")
  lcdDisplay.SetCursor(0, 1)       'set the cursor at the begin of second line
  lcdDisplay.Write("red ")        'write "red " in the second line
  pinLEDYellow.DigitalWrite(False) 'switch OFF LED Yellow
  pinLEDRed.DigitalWrite(True)    'switch ON LED Red
  TimerYellow.Enabled = False      'disable Timer TimerYellow
End Sub

```

5.10 PulseWidthMeter

We reuse the LCDDisplay project.

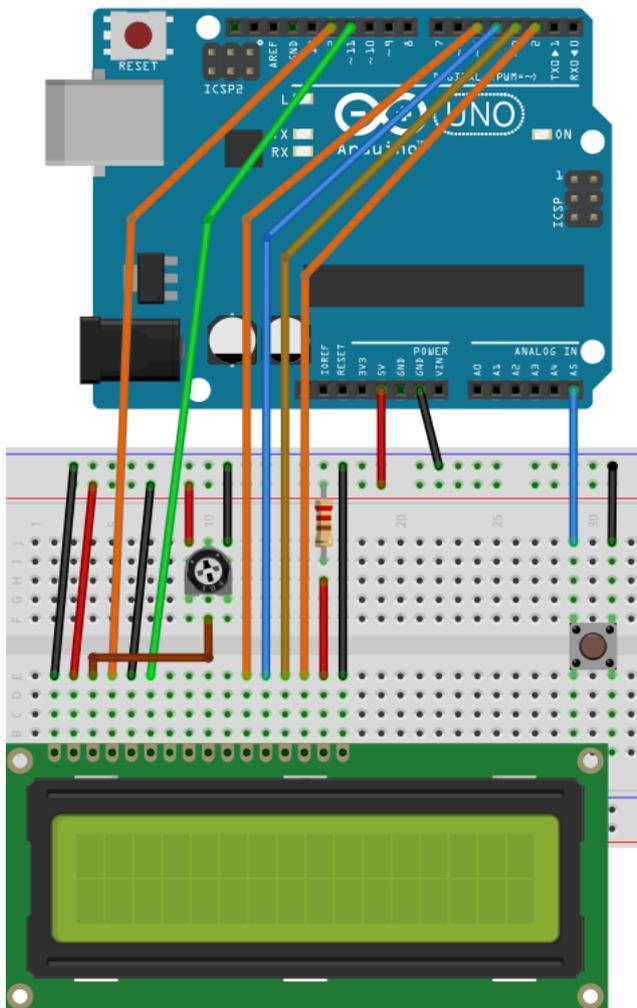
Make a new folder PulseWidthMeter, copy there all the files of the LCDDisplay project and rename the *LCDDisplay.xxx* files to *PulseWidthMeter.xxx*.

The project PulseWidthMeter.b4r is available in the SourceCode folder.

For this project we need the rLiquidCrystal library.

In the Libraries Tab select the rLiquidCrystal library.

5.10.1 Sketch



Material:

- 1 pushbutton switch
- 1 potentiometer
- 1 LCD display
- 1 220 Ω resistor

It's the same as the LCDDisplay circuit, without the three LEDs.

You can even leave the LEDs, they are simply not used.

5.10.2 Code

Sub Process_Globals

```
Public Serial1 As Serial

Public pinButton As Pin           'pin for the button
Public lcdDisplay As LiquidCrystal

Public TimeBegin, PulseWidth As ULong 'used to measure the time
End Sub
```

We leave the definition of `pinButton` and `lcdDisplay` and remove the rest. We add two variables `TimeBegin` and `pinButton` used to measure the time.

Private Sub AppStart

```
Serial1.Initialize(115200)

'Using the internal pull up resistor to prevent the pin from floating.
pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
pinButton.AddListener("pinButton_StateChanged")

'initialize the display(RS As Byte, RW As Byte, Enable As Byte
'RS pin > Arduino digital pin 12
'RW pin > 255 means not used
'Enable pin > Arduino digital pin 11
'DataPins: Arduino digital pins 5, 4, 3, 2
lcdDisplay.Initialize(12, 255, 11, Array As Byte(5, 4, 3, 2))
lcdDisplay.Begin(16, 2) 'set the display type 2 * 16 characters
lcdDisplay.Write("Pulse width [ms]") 'write "Pulse width" in the first line
End Sub
```

Nothing new.

Private Sub pinButton_StateChanged (State As Boolean)

```
' Log("State: ", State) 'Log the State value
If State = False Then 'if State = False, button down
    TimeBegin = Millis
Else
    'calculate the number of milliseconds since button down
    PulseWidth = Millis - TimeBegin
    lcdDisplay.SetCursor(0, 1) 'set the cursor at the begin of the second line
    lcdDisplay.Write(" ") 'clear the text
    lcdDisplay.SetCursor(0, 1) 'set the cursor at the begin of the second line
    lcdDisplay.Write(PulseWidth) 'write the number of milli seconds
End If
End Sub
```

When we press the button, we measure the time since the last restart.

When we release the button, we measure the number of milli seconds since button down and display the value on the display.

5.11 ObjectArrays.b4r

This project shows how to use object arrays.

The circuit has 4 LEDs which are switched ON and OFF with a timer.

The LEDs can be switched ON or OFF with the right pushbutton switch.

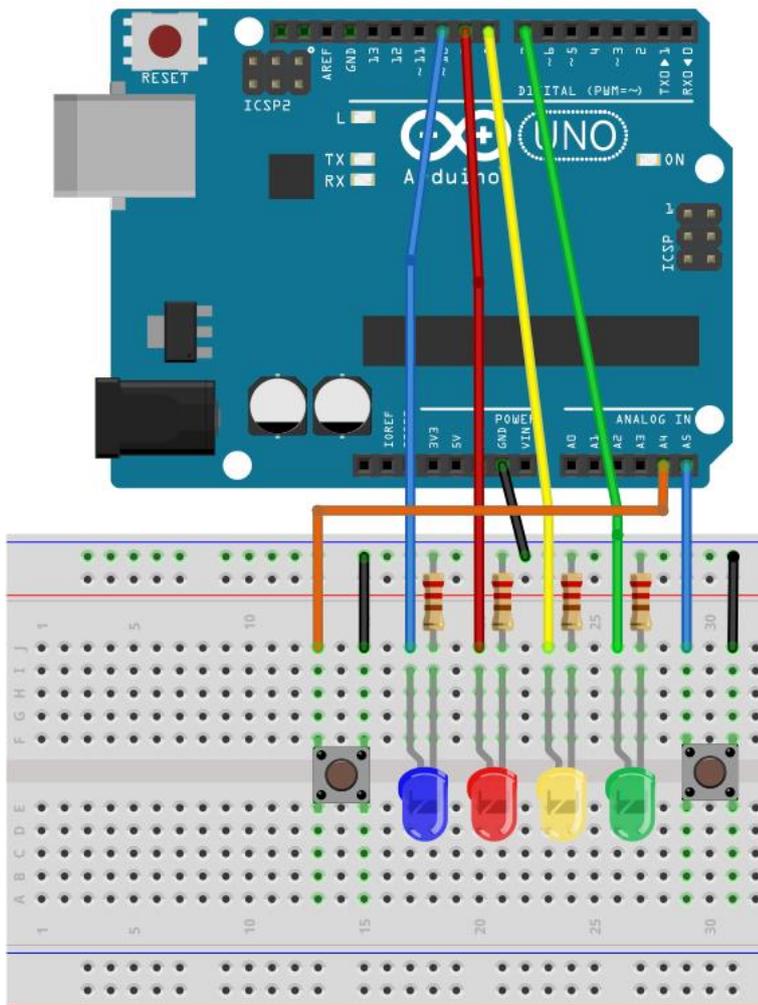
Two display modes selected with left pushbutton switch:

- All LEDs are OFF, only the current LED is ON.
- All LEDs are ON, only the current LED is OFF.

We use two arrays of pins to manage the two pushbuttons and the four LEDs.

The project ObjectArrays.b4r is available in the SourceCode folder.

5.11.1 Sketch



The circuit is based on the TrafficLight circuit.

We add another pushbutton switch and a blue LED with its resistor.

Material:

- 2 pushbutton switches
- 1 green Led
- 1 yellow Led
- 1 red led
- 1 blue led
- 4 220 Ω resistors

The right pushbutton switch has one pin connected to GND and the other pin connected to the analog pin **A5**.

The left pushbutton switch has one pin connected to GND and the other pin connected to the analog pin **A4**.

The 4 LEDs have their cathodes (-) connected to GND via a 220 Ω resistor

The LED anodes (+) are connected:

- Green to digital pin **7**.
- Yellow to digital pin **8**.
- Red to digital pin **9**.
- Blue to digital pin **10**.

5.11.2 Code

Sub Process_Globals

```

Public Serial1 As Serial

Public NbLEDs = 4 As Int           'Number of Leds
Public pinLEDs(4) As Pin          'Pins for the Leds
Public NbButtons = 2 As Int       'Number of buttons
Public pinButtons(2) As Pin      'Pins for the buttons

Public TimerLEDs As Timer
Public LightON = False As Boolean 'lights ON or OFF, ON = True
Public LEDIndex = 0 As Int        'index of current LED
Public AllLedsON = False As Boolean 'lighting type
Private BounceTime(2) As ULong
Private BounceDelay = 10 As ULong
End Sub

```

We declare the objects and variables.

Instead of declaring 4 individual pins for the LEDs like:

```
Public pinLEDGreen, pinLEDYellow, pinLEDRed, pinLEDBlue As Pin
```

We declare an array of 4 pins.

```
Public pinLEDs(4) As Pin
```

The same for the 2 pushbutton switches.

We use LightON to switch all the LEDs ON or OFF.

We use LEDIndex for the current LED to switch it ON or OFF depending on the lighting type AllLedsON.

We use AllLedsON for the lighting type.

- AllLedsON = False All LEDs switched OFF only the current LED is switched ON.
- AllLedsON = True All LEDs switched ON only the current LED is switched OFF.

In B4R this code doesn't work:

```

Public NbLEDs = 4 As Int           'Number of Leds
Public pinLEDs(NbLEDs) As Pin     'Pins for the Leds

```

In `Public pinLEDs(4) As Pin` the value must be a numeric literal, not a variable.

`Public pinLEDs(NbLEDs) As Pin` throws this error:

The size of non-primitive arrays must be a numeric literal.

Private Sub AppStart

```

Private i As Int

Serial1.Initialize(115200)

TimerLEDs.Initialize("TimerLEDs_Tick", 500)

'Initialize the buttons and add the StateChange event
'Analog pins A4, A5
'Analog pins A0 = 14, A1 = 15, A2 = 16, A3 = 17, A4 = 18, A5 = 19
'in our case (i + 18), 18 and 19, therefore A4 and A5
For i = 0 To NbButtons - 1
    'Using the internal pull up resistor to prevent the pin from floating.
    pinButtons(i).Initialize( i + 18, pinButtons(i).MODE_INPUT_PULLUP)
    pinButtons(i).AddListener("pinButtons_StateChanged")
Next

'Initialize the LEDs
'Digital pins 7, 8, 9, 10
For i = 0 To NbLEDs - 1
    pinLEDs(i).Initialize(i + 7, pinLEDs(i).MODE_OUTPUT)
Next
End Sub

```

Instead of initializing the pushbutton switches like:

```

pinButton1.Initialize(pinButton1.A4, pinButton1.MODE_INPUT_PULLUP)
pinButton1.AddListener("pinButtons_StateChanged")
pinButton2.Initialize(pinButton2.A5, pinButtons2.MODE_INPUT_PULLUP)
pinButton2.AddListener("pinButtons_StateChanged")

```

We use a For / Next loop:

```

For i = 0 To NbButtons - 1
    'Using the internal pull up resistor to prevent the pin from floating.
    pinButtons(i).Initialize( i + 18, pinButtons(i).MODE_INPUT_PULLUP)
    pinButtons(i).AddListener("pinButtons_StateChanged")
Next

```

The analog pins can be addressed either with:

pin.A0 or 18, pin.A1 or 19, pin.A2 or 20 etc.

The same for the LEDs, instead of:

```

pinLEDGreen.Initialize(7, pinLEDGreen.MODE_OUTPUT)
pinLEDYellow.Initialize(8, pinLEDYellow.MODE_OUTPUT)
pinLEDRed.Initialize(9, pinLEDRed.MODE_OUTPUT)
pinLEDBlue.Initialize(10, pinLEDBlue.MODE_OUTPUT)

```

We use also a For / Next loop:

```

For i = 0 To NbLEDs - 1
    pinLEDs(i).Initialize(i + 7, pinLEDs(i).MODE_OUTPUT)
Next

```

```

Private Sub pinButtons_StateChanged (State As Boolean)
  Private p As Pin

  Log("State: ", State)           'Log the State value

  p = Sender                       'get the Pin which raised the event
  Log("Pin Number: ", p.PinNumber) 'Log the Pin Number

  Select p.PinNumber
  Case 18 'Light type
    If State = False Then           'if State = False, button down
      If Millis - BounceTime(0) < BounceDelay Then
        Return
      Else
        AllLedsON = Not(AllLedsON) 'change the state of AllLedsON
        BounceTime(0) = Millis
        If LightON = True Then      'if LightOn = False
          For i = 0 To NbLEDs - 1
            'switch OFF or ON all LEDs depending on AllLedsON
            pinLEDs(i).DigitalWrite(AllLedsON)
          Next
          'switch ON or OFF the current LED depending on AllLedsON
          pinLEDs(LEDIndex).DigitalWrite(Not(AllLedsON))
        End If
      End If
    End If
  Case 19 'Lights ON / OFF
    If State = False Then           'if State = False, button down
      If Millis - BounceTime(1) < BounceDelay Then
        Return
      Else
        LightON = Not(LightON)      'change the value of LightON
        Log("Light: ", LightON)     'Log the LightOn value
        BounceTime(1) = Millis

        TimerLEDs.Enabled = LightON 'enable or disable TimerLEDs Timer

        If LightON = False Then      'if LightOn = False, lights OFF
          For i = 0 To NbLEDs - 1
            pinLEDs(i).DigitalWrite(False) 'switch OFF all LEDs
          Next
        Else
          For i = 0 To NbLEDs - 1
            pinLEDs(i).DigitalWrite(AllLedsON) 'switch OFF or ON all LEDs
          Next
          'switch ON or OFF the current LED
          pinLEDs(LEDIndex).DigitalWrite(Not(AllLedsON))
        End If
      End If
    End If
  End Select
End Sub

```

We have only one StateChanged event routine therefore we need to know which pushbutton switch raised the event.

We use, like in the other B4x languages, the Sender object.

```
Private p As Pin  
p = Sender
```

In the other B4x languages we can use the Tag property of the object to know which one raised the event. In B4R the Tag property doesn't exist.

We use the PinNumber property.

```
Select p.PinNumber
```

And Case 18 for the lighting type pushbutton switch.

And Case 19 for the light ON/OFF pushbutton switch.

Using an array for the LEDs and variables for the lighting type and the current active LED is a big advantage because the code becomes much shorter, but a bit more difficult to read.

```

If LightON = True Then          'if LightOn = False
  For i = 0 To NbLEDs - 1
    'switch OFF or ON all LEDs depending on AllLedsON
    pinLEDs(i).DigitalWrite(AllLedsON)
  Next
  'switch ON or OFF the current LED depending on AllLedsON
  pinLEDs(LEDIndex).DigitalWrite(Not(AllLedsON))
End If

```

The code above is much simpler than the code below we should have written without the arrays:

```

If LightON = True Then
  If AllLedsON = False Then
    'switch OFF all LEDs
    pinLedGreen.DigitalWrite(False)
    pinLedYellow.DigitalWrite(False)
    pinLedRed.DigitalWrite(False)
    pinLedBlue.DigitalWrite(False)
    Select LEDIndex)
    Case 0
      pinLedGreen.DigitalWrite(True)
    Case 1
      pinLedYellow.DigitalWrite(True)
    Case 2
      pinLedRed.DigitalWrite(True)
    Case 3
      pinLedBlue.DigitalWrite(True)
    End Select
  Else
    'switch ON all LEDs
    pinLedGreen.DigitalWrite(True)
    pinLedYellow.DigitalWrite(True)
    pinLedRed.DigitalWrite(True)
    pinLedBlue.DigitalWrite(True)
    Select LEDIndex)
    Case 0
      pinLedGreen.DigitalWrite(False)
    Case 1
      pinLedYellow.DigitalWrite(False)
    Case 2
      pinLedRed.DigitalWrite(False)
    Case 3
      pinLedBlue.DigitalWrite(False)
    End Select
  End If
End If

```

```

Private Sub TimerLEDs_Tick
  pinLEDs(LEDIndex).DigitalWrite(AllLedsON) 'switch OFF or ON the current LED
  'increment LedIndex by 1 and set it to 0 if LEDIndex = NbLEDs
  LEDIndex = (LEDIndex + 1) Mod NbLEDs
  pinLEDs(LEDIndex).DigitalWrite(Not(AllLedsON)) 'switch ON or OFF the current LED
End Sub

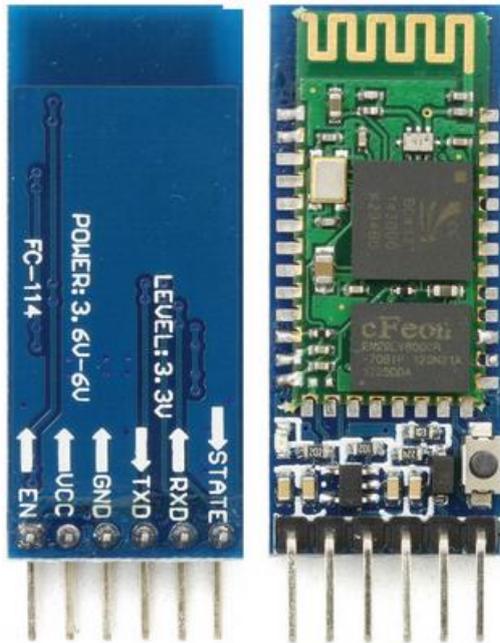
```

This code is also quite simple.

6 HC-05 Bluetooth

The HC-05 module allows Bluetooth communication between an Arduino or similar boards and any other device.

Attention: The HC-05 module works with 3.3V and not with 5V!



Pins:

- STATE Shows the state of the HC-05, the STATE pin is LOW when the HC-05 is not connected and HIGH when the HC-05 is connected.
- RXD Serial communication RX.
- TXD Serial communication TX.
- GND Power supply ground.
- VCC Power supply 3.6 to 6V.
- EN Used to program the HC-05, we don't use it.

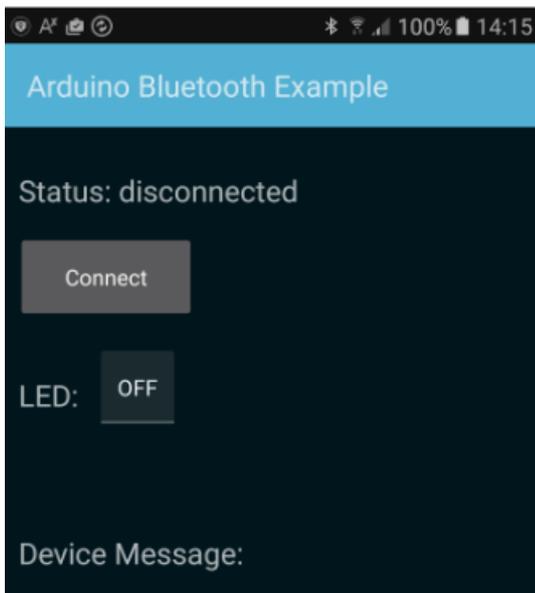
All the example programs using the HC-05 Bluetooth module use a same class for the communication between the Arduino and the smartphone.

6.1 Program HC05LedOnOff.b4r

With this first program we use an Android device to switch ON and OFF a LED via a HC-05 board and an Arduino UNO.

The project HC05LedOnOff is available in the SourceCode\HC05 folder. It is based on [Erels project](#) in the forum, a little bit modified.

The project includes the B4R program managing the Arduino UNO and the HC-05 board. And a B4A program to communicate between an Android device and the Arduino.



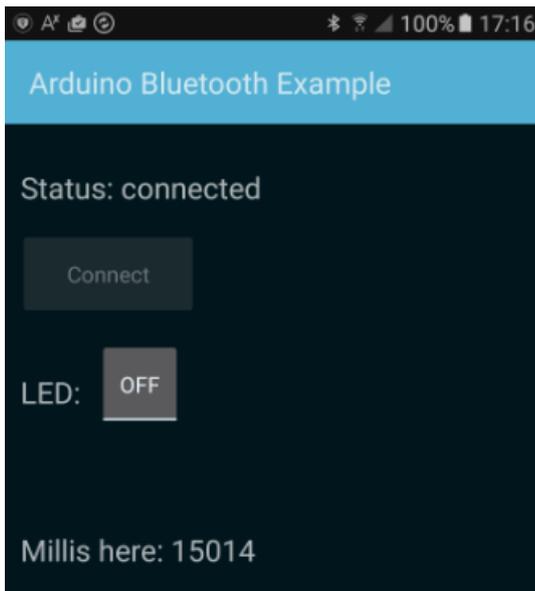
Display of the status.

Button to connect the two devices.

A Togglebutton to switch the LED ON or OFF.

Message returned from the Arduino.

The Arduino UNO sends every second a message with the milliseconds elapsed since its connection.



When we click on  the first time we are asked if we want to pair the two devices. Enter "1234" for the PIN code.

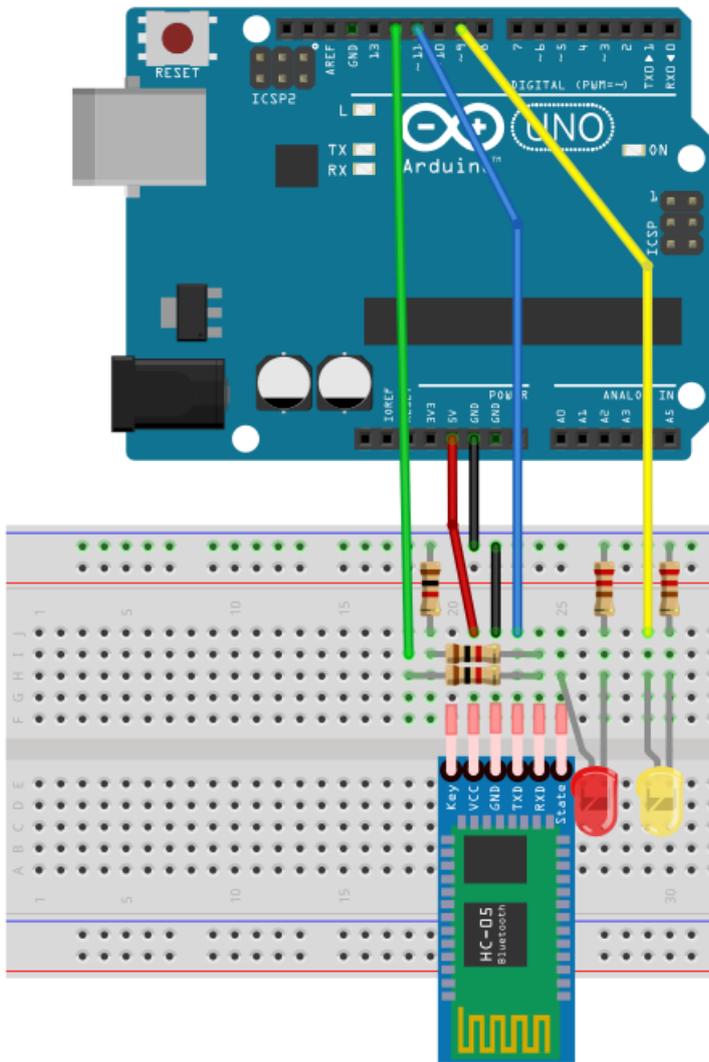
Then the Android device tries to connect.

When the connection is established, the Status changes, the Connect button is disabled and the Togglebutton is enabled.

And we get messages from the Arduino.

The Arduino UNO sends every second a message with the milliseconds elapsed since its connection.

6.1.1 Sketch



We use:

- 1 Arduino UNO
- 1 HC-05 board
- 1 yellow LED
- 1 red LED, shows the HC-05 state
- 3 $1\text{k}\Omega$ resistors for the voltage divider
- 2 220Ω resistors for the LEDs

We connect_

- The GND pin of the UNO to the GND line of the breadboard.
- The 5V pin of the UNO to the VIN pin of the HC-05.
- The GND pin of the HC-05 to the GND line of the breadboard.
- Pin D11 of the UNO to the TXD pin of the HC-05.
- Pin D12 of the UNO to the RXD pin of the HC-05 via the voltage divider.
- Pin D9 of the UNO to the anode of the yellow LED.
- One 220Ω resistor between the cathode of the yellow LED and the GND line of the breadboard.
- The anode of the red LED to the STATE pin of the HC-05.
- One 220Ω resistor between the cathode of the red LED and the GND line of the breadboard.

Because of the different voltage levels between the Arduino UNO (5V) and the HC-05 (3.3V) we need to adapt signal levels.

The power supply for the HC-05 accepts values between 3.6 and 6V, so we provide it from the Arduino 5V pin.

The TXD pin of the HC-05 can be directly connected to the RX wire, pin D11 of the Arduino in our case. The 3.3V level is sufficient to drive an Arduino digital pin.

To connect the TX wire of the Arduino (pin D12 in our case) we need a voltage divider to convert the 5V down to 3.3V.

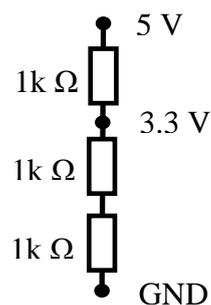
The voltage divider:

We use three $1\text{k}\Omega$ resistors.

We could also have used one $1\text{k}\Omega$ resistor and one $2\text{k}\Omega$ resistor.

But I did not have a $2\text{k}\Omega$ resistor.

So, I use two $1\text{k}\Omega$ resistors in series.



6.1.2 Code

We need two programs, one for the Arduino UNO and one for the Android device.

6.1.2.1 B4R Arduino UNO

We need two libraries:

- rRandomAccessFile
- rSoftwareSerial

```
Sub Process_Globals
  Public Serial1 As Serial
  Private SoftwareSerial1 As SoftwareSerial
  Private astream As AsyncStreams
  Private YellowLED As Pin
  Private Timer1 As Timer
End Sub
```

We declare the different variables.

We initialize:

- The yellow LED as digital pin D5
- The software serial interface with the digital pin D11 for the RX signal and the digital pin D12 for the TX signal. We use a software serial interface because the ‘standard’ RX and TX lines are used for the communication between B4R and the Arduino UNO.
- The AsyncStream to send the messages.
- The Timer for the message timing.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")
  YellowLED.Initialize(5, YellowLED.MODE_OUTPUT)
  SoftwareSerial1.Initialize(9600, 11, 12) 'software serial port on pins 12 and 11
  astream.Initialize(SoftwareSerial1.Stream, "astream_NewData", Null)
  Timer1.Initialize("timer1_Tick", 1000)
  Timer1.Enabled = True
End Sub
```

In every tick we send the message “Millis here:” plus the milliseconds elapsed since the connection of the Arduino.

```
Sub Timer1_Tick
  astream.Write("Millis here: ".GetBytes)
  astream.Write(NumberFormat(Millis, 0, 0).GetBytes)
  astream.Write(Array As Byte(10)) 'end of line character. AsyncStreamsText will cut
  the message here
End Sub
```

We get the message from the Android device and set the value to the yellow LED.

```
Sub AStream_NewData (Buffer() As Byte)
  Dim value As Boolean = Buffer(0) = 1
  YellowLED.DigitalWrite(value)
End Sub
```

6.1.2.2 B4A Android

Needs two libraries:

- RandomAccessFile
- Serial

And one Class:

- AsyncStreamsText

All the communication is managed in the Starter service module.

Sub Globals

```
Private tbtLED As ToggleButton
Private lblStatus As Label
Private btnConnect As Button
Private lblMessage As Label
Private ProgressBar1 As ProgressBar
```

```
Private Status As String
```

End Sub

We use:

- A ToggleButton to switch ON or OFF the LED.
- A Label displaying the connecting status.
- A Button to connect the device to the Arduino.
- A Label to display the message sent by the Arduino.
- A ProgressBar, shown during the connection.
- A String variable, containing the connecting status.

```
Sub Activity_Create(FirstTime As Boolean)
```

```
Activity.LoadLayout("Main")
```

End Sub

We load the layout.

```
Sub Activity_Resume
```

```
SetState
```

End Sub

We set the connecting state.

```
Sub Activity_Pause (UserClosed As Boolean)
```

```
If UserClosed = True And Status = "connected" Then
    CallSub2(Starter, "SendMessage", Array As Byte(0))
```

```
End If
```

End Sub

We switch off the LED when the user leaves the program.

```
Public Sub SetState
    tbtLED.Enabled = Starter.connected
    btnConnect.Enabled = Not(Starter.connected)
    ProgressBar1.Visible = Starter.connecting
    If Starter.Connected Then
        Status = "connected"
    Else If Starter.TryToConnect Then
        Status = "trying to connect..."
    Else If Starter.Connecting Then
        Status = "HC-05 found connecting..."
    Else
        Status = "disconnected"
    End If
    lblStatus.Text = $"Status: ${Status}"$
End Sub
```

We display the connecting status text in lblStatus.

```
Public Sub MessageFromDevice(msg As String)
    lblMessage.Text = msg
End Sub
```

We display the message set by the Arduino.

```
Sub tbtLED_CheckedChange(Checked As Boolean)
    Dim b As Byte
    If Checked = True Then
        b = 1
    Else
        b = 0
    End If
    CallSub2(Starter, "SendMessage", Array As Byte(b))
End Sub
```

Depending on the state of the ToggleButton we send either 1 (True) or 0 (False) to the Ardiono.

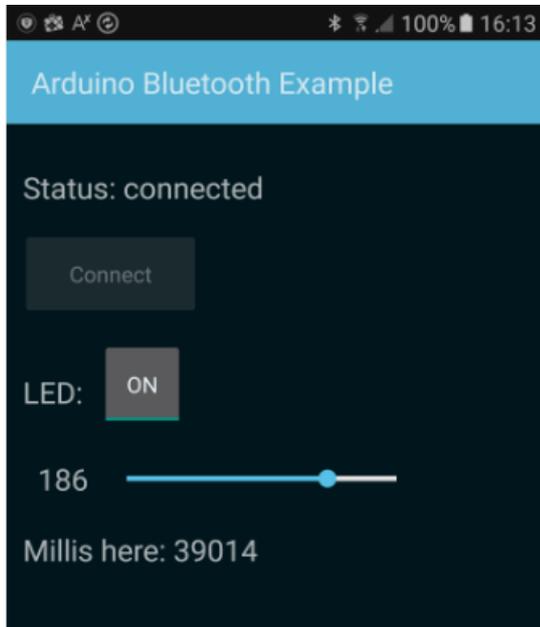
```
Sub btnConnect_Click
    CallSub(Starter, "Connect")
End Sub
```

When the Button btnConnect is clicked we start the connection with the Arduino.

6.2 Program HC05LightDimmer

This project is almost the same as the HC05LedOnOff, the difference is that we can modulate the light intensity with a Seekbar on the Android device.

The project HC05LightDimmer is available in the SourceCode\HC05 folder.



Display of the status.

Button to connect the two devices.

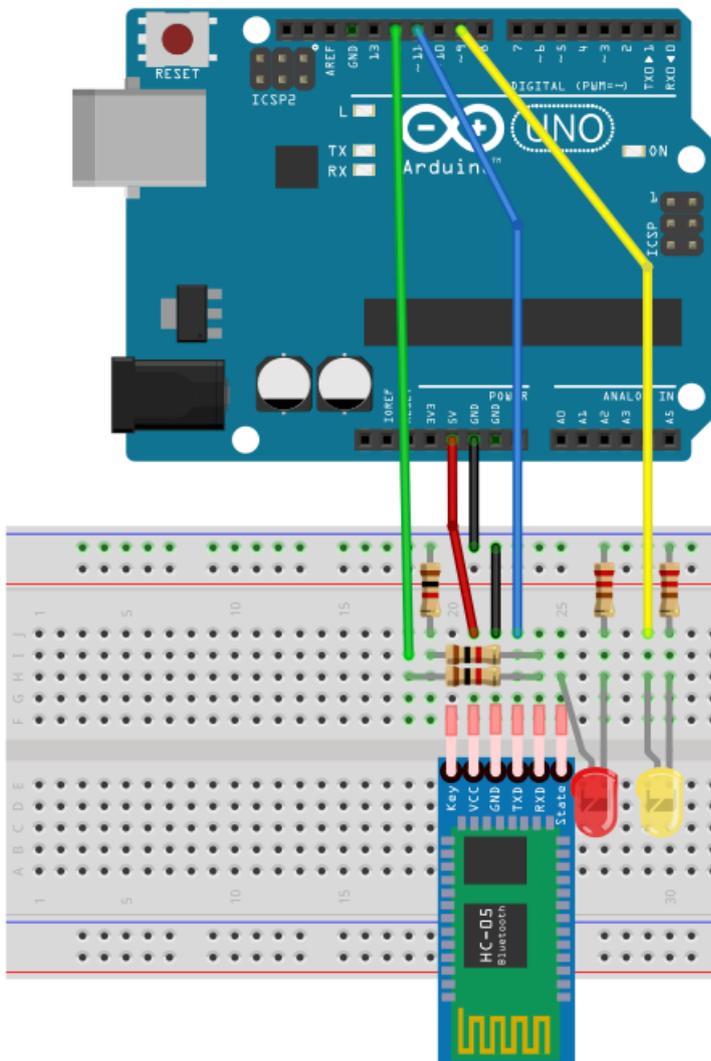
A Togglebutton to switch the LED ON or OFF.

Slider to control the LED brightness.

Message returned from the Arduino.

The Arduino UNO sends every second a message with the milliseconds elapsed since its connection.

6.2.1 Sketch



Same as the HC05LedOnOff program.

We use:

- 1 Arduino UNO
- 1 HC-05 board
- 1 yellow LED
- 1 red LED, shows the HC-05 state
- 3 1k Ω resistors for the voltage divider
- 2 220 Ω resistors for the LEDs

We connect_

- The GND pin of the UNO to the GND line of the breadboard.
- The 5V pin of the UNO to the VIN pin of the HC-05.
- The GND pin of the HC-05 to the GND line of the breadboard.
- Pin D11 of the UNO to the TXD pin of the HC-05.
- Pin D12 of the UNO to the RXD pin of the HC-05 via the voltage divider.
- Pin D9 of the UNO to the anode of the yellow LED.
- One 220 Ω resistor between the cathode of the yellow LED and the GND line of the breadboard.
- The anode of the red LED to the STATE pin of the HC-05.
- One 220 Ω resistor between the cathode of the red LED and the GND line of the breadboard.

Because of the different voltage levels between the Arduino UNO (5V) and the HC-05 (3.3V) we need to adapt signal levels.

The power supply for the HC-05 accepts values between 3.6 and 6V, so we provide it from the Arduino 5V pin.

The TXD pin of the HC-05 can be directly connected to the RX wire, pin D11 of the Arduino in our case. The 3.3V level is sufficient to drive an Arduino digital pin.

To connect the TX wire of the Arduino (pin D12 in our case) we need a voltage divider to convert the 5V down to 3.3V.

6.2.2 Code

6.2.2.1 B4R Arduino

We need two libraries:

- rRandomAccessFile
- rSoftwareSerial

This project is almost the same as the HC05LedOnOff program, only the differences is shown below.

HC05LightDimmer

```
Sub AStream_NewData (Buffer() As Byte)
    Dim value As UInt = Buffer(0)
    YellowLED.AnalogWrite(value)
End Sub
```

HC05LedOnOff

```
Sub AStream_NewData (Buffer() As Byte)
    Dim value As Boolean = Buffer(0) = 1
    YellowLED.DigitalWrite(value)
End Sub
```

In these routines we get the data sent by the connected device.

In HC05LightDimmer we get an integer value converted into a byte.

In HC05LedOnOff we get a Boolean value converted into a byte.

6.2.2.2 B4A Android

The B4A project is almost the same, here too, only the differences are shown below.

HC05LightDimmer

```
Sub Globals
    Private tbtLED As ToggleButton
    Private lblStatus As Label
    Private btnConnect As Button
    Private lblMessage As Label
    Private ProgressBar1 As ProgressBar
    Private skbDimmer As SeekBar
    Private lblDimmer As Label

    Private Status As String
End Sub
```

HC05LedOnOff

```
Sub Globals
    Private tbtLED As ToggleButton
    Private lblStatus As Label
    Private btnConnect As Button
    Private lblMessage As Label
    Private ProgressBar1 As ProgressBar

    Private Status As String
End Sub
```

Just two new objects, a Seekbar skbDimmer and a Label lblDimmer.

And a new routine for the Seekbar:

```
Sub skbDimmer_ValueChanged (Value As Int, UserChanged As Boolean)
    lblDimmer.Text = Value
    If tbtLED.Checked = True Then
        Dim b As Byte
        b = Value
        CallSub2(Starter, "SendMessage", Array As Byte(b))
    End If
End Sub
```

6.3 Program HC05DataLogger.b4r

With this program we use an Android device to get data from an Arduino UNO via a HC-05 board.

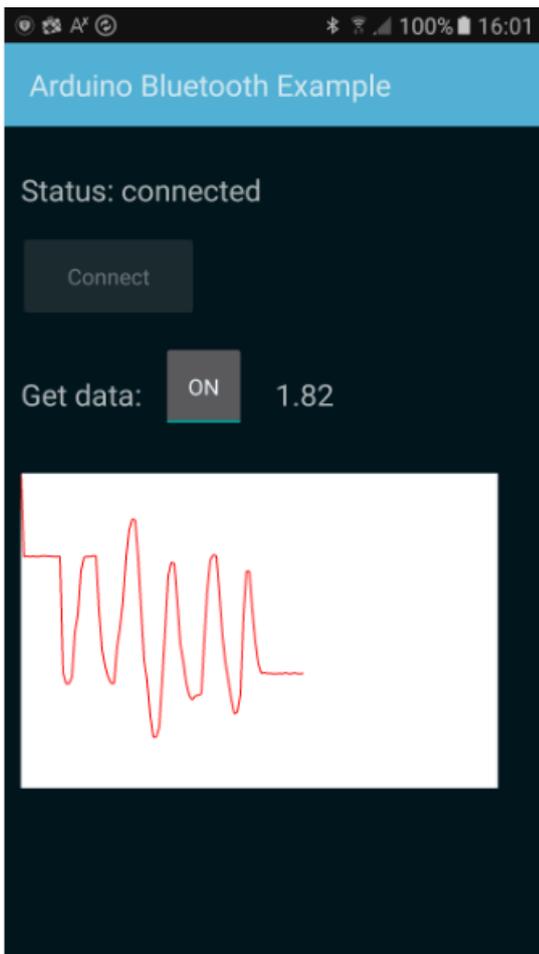
The data on the Arduino is generated with a potentiometer.

The project HC05DataLogger is available in the SourceCode\HC05 folder.

It is based on [Erels project](#) in the forum, a little bit modified.

The project includes the B4R program managing the Arduino UNO and the HC-05 board.

And a B4A program to communicate between the Android device and the Arduino.



Display of the status.

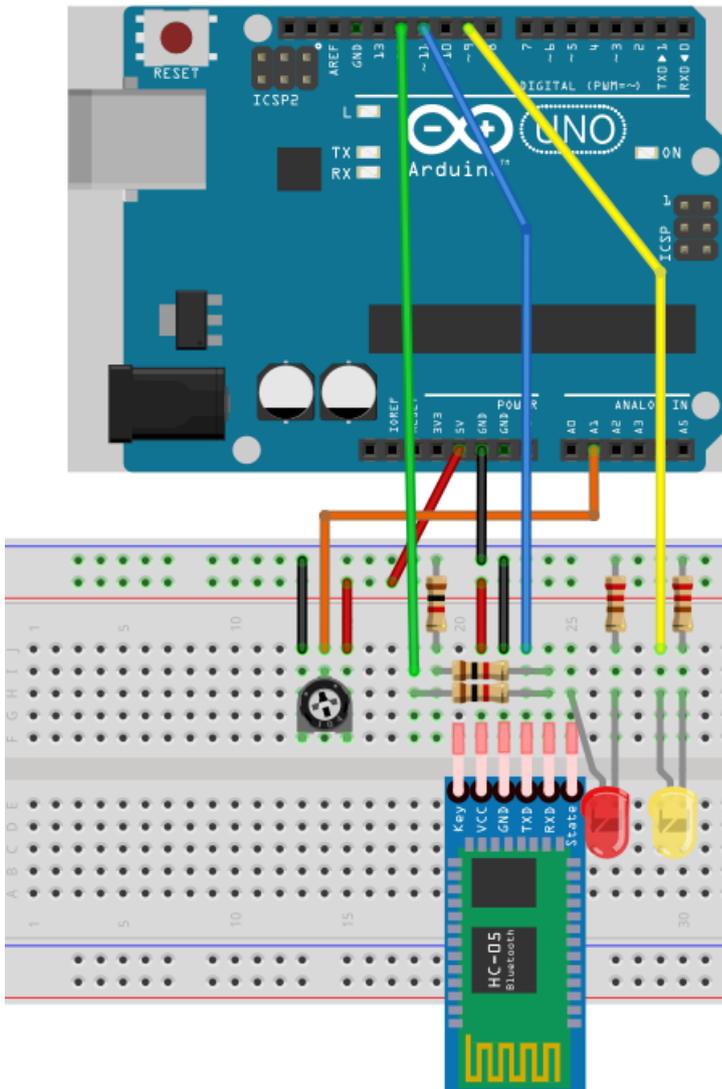
Button to connect the two devices.

A Togglebutton to switch ON or OFF the data stream.
Display of the last data.

Display of the transmitted data.

The Arduino UNO sends every 100ms the value of the potentiometer.

6.3.1 Sketch



We use:

- 1 Arduino UNO
- 1 HC-05 board
- 1 yellow LED
- 1 red LED, shows the HC-05 state
- 3 $1\text{ k}\Omega$ resistors for the voltage divider
- 2 $220\ \Omega$ resistors for the LEDs

We connect_

- The GND pin of the UNO the GND of the breadboard.
- The 5V pin of the UNO to the VIN pin of the HC-05.
- The GND pin of the HC-05 to the GND line of the breadboard.
- Pin D11 of the UNO to the TXD pin of the HC-05.
- Pin D12 of the UNO to the RXD pin of the HC-05 via the voltage divider.
- Pin D9 of the UNO to the anode of the yellow LED.
- One $220\ \Omega$ resistor between the cathode of the yellow LED and the GND line of the breadboard.
- The anode of the red LED to the STATE pin of the HC-05.
- One $220\ \Omega$ resistor between the cathode of the red LED and the GND line of the breadboard.
- One potentiometer connected to

GND and 5V, the slider is connected to pin A1.

Because of the different voltage levels between the Arduino UNO (5V) and the HC-05 (3.3V) we need to adapt signal levels.

The power supply for the HC-05 accepts values between 3.6 and 6V, so we provide it from the Arduino 5V pin.

The TXD pin of the HC-05 can be directly connected to the RX wire, pin D11 of the Arduino in our case. The 3.3V level is sufficient to drive an Arduino digital pin.

To connect the TX wire of the Arduino (pin D12 in our case) we need a voltage divider to convert the 5V down to 3.3V.

6.3.2 Code

6.3.2.1 B4R Arduino

```
Sub Process_Globals
  Public Serial1 As Serial
  Private SoftwareSerial1 As SoftwareSerial
  Private astream As AsyncStreams
  Private Timer1 As Timer
  Private pinData, pinLED As Pin
  Private Send As Boolean
End Sub
```

We use:

- 1 Timer, to send periodically the position of the potentiometer slider.
- 2 pins
 - 1 analog pin, pinData, for the potentiometer slider.
 - 1 digital pin, showing the Send state.
- 1 variable, Send, data stream enabled or disabled, state of the Togglebutton in B4A.

```
Private Sub AppStart
  Serial1.Initialize(115200)
  Log("AppStart")
  SoftwareSerial1.Initialize(9600, 11, 12) 'software serial port on pins 12 and 11
  astream.Initialize(SoftwareSerial1.Stream, "astream_NewData", Null)
  pinData.Initialize(pinData.A1, pinData.MODE_INPUT)
  pinLED.Initialize(5, pinLED.MODE_OUTPUT)
  Timer1.Initialize("Timer1_Tick", 100)
End Sub
```

We initialize:

- The serial port, like in the other HC-05 programs.
- The two pins.
- The Timer, with a period of 100ms.

```
Private Sub Timer1_Tick
  'sends the message, values from analog pinData, analog pin A1
  astream.Write(NumberFormat(pinData.AnalogRead, 0, 0).GetBytes)
  astream.Write(Array As Byte(10))
  'end of line character. AsyncStreamsText will cut the message here
End Sub
```

The Timer sends every 100ms the value of the potentiometer slider.

We need to convert the analog pin Byte value to a String.

We send a LF character which acts as an end of message character.

```
Private Sub AStream_NewData (Buffer() As Byte)
  'gets the Bluetooth data
  Send = Buffer(0) = 1 'data received, one byte, boolean value
  pinLED.DigitalWrite(Send) 'sets the LED
  Timer1.Enabled = Send 'enables or disables the Timer
End Sub
```

Gets the data from the connected device, state of ToggleButton in our case.

6.3.2.2 B4A Android

All the communication is managed in the Starter service module.

Variable declaration:

```
Sub Globals
  Private tbtLED As ToggleButton
  Private lblStatus As Label
  Private btnConnect As Button
  Private lblMessage As Label
  Private ProgressBar1 As ProgressBar
  Private pnlDisplay As Panel
  Private cvsDisplay As Canvas
  Private Value As Double
  Private Counter = 0 As Int
  Private XStep = 2dip As Int
  Private Scale As Double
  Private x0, y0, x1, y1 As Int
  Private rectDisplay, rectVal As Rect
End Sub
```

Initialize the canvas and drawing:

```
Sub Activity_Create(FirstTime As Boolean)
  Activity.LoadLayout("Main")

  cvsDisplay.Initialize(pnlDisplay)
  rectDisplay.Initialize(0, 0, pnlDisplay.Width, pnlDisplay.Height)
  DrawInit 'initializes the drawing
End Sub
```

Stops sending data from the Arduino when the program is left:

```
Sub Activity_Pause (UserClosed As Boolean)
  If UserClosed = True And lblStatus.Text = "connected" Then
    CallSub2(Starter, "SendMessage", Array As Byte(0))
  End If
End Sub
```

Connection state displays routine, also called from the Starter service depending on the state.

```
Public Sub SetState
  tbtLED.Enabled = Starter.connected
  btnConnect.Enabled = Not(Starter.connected)
  ProgressBar1.Visible = Starter.connecting
  Dim status As String
  If Starter.Connected Then
    status = "connected"
  Else If Starter.TryToConnect Then
    status = "trying to connect..."
  Else If Starter.Connecting Then
    status = "HC-05 found connecting..."
  Else
    status = "disconnected"
  End If
  lblStatus.Text = $"Status: {status}"$
End Sub
```

Reading the messages from the Arduino and display of the value.

This routine is called from the Starter service when a new data is sent from the Arduino.

```
Public Sub MessageFromDevice(msg As String)
    Private Val As Double
    Val = msg
    Value = 5 * Val / 1024
    lblMessage.Text = NumberFormat(Value, 0, 2)

    DrawValue
End Sub
```

Sends a message to the Arduino to start or stop sending data when pressing the tbtLED ToggleButton.

```
Sub tbtLED_CheckedChange(Checked As Boolean)
    Dim b As Byte
    If Checked = True Then
        b = 1
    Else
        b = 0
        lblMessage.Text = "No data"
    End If
    CallSub2(Starter, "SendMessage", Array As Byte(b))
End Sub
```

Draws a new value to the diagram.

```
Private Sub DrawValue
    Counter = Counter + XStep
    x1 = Counter
    y1 = pnlDisplay.Height - Value * Scale
    rectVal.Initialize(x0, 0, x1, pnlDisplay.Height)
    cvsDisplay.DrawLine(x0, y0, x1, y1, Colors.Red, 1dip)
    pnlDisplay.Invalidate2(rectVal)
    x0 = x1
    y0 = y1
    'clears the diagram when filled
    If Counter >= pnlDisplay.Width Then
        x0 = 0
        Counter = 0
        cvsDisplay.DrawRect(rectDisplay, Colors.White, True, 1)
        pnlDisplay.Invalidate
    End If
End Sub
```

6.4 Program HC-05 DCMotor

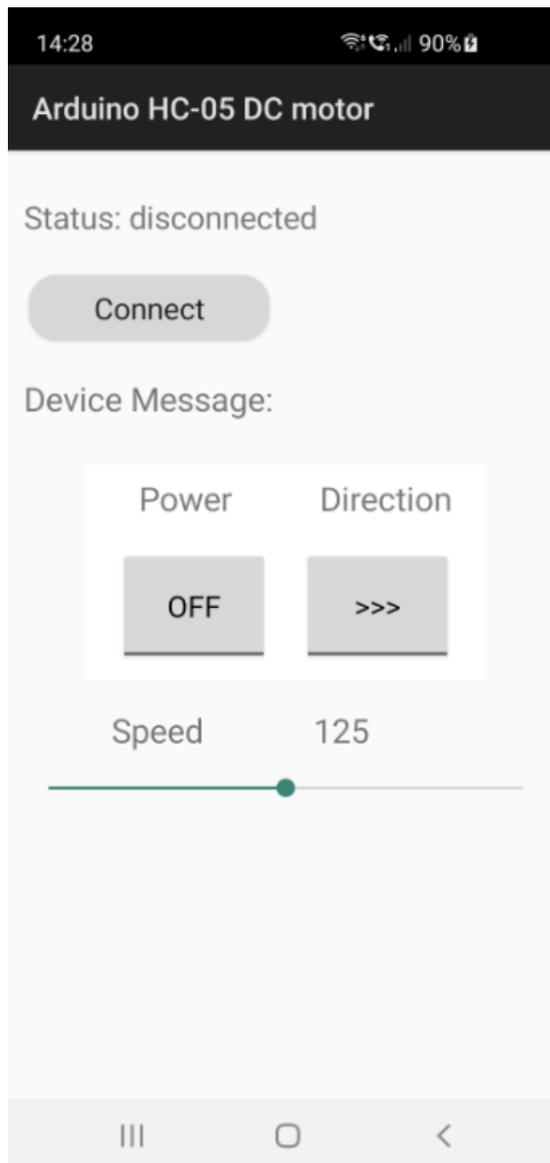
With this program we use an Android device to control a DC motor.

This project is like the DCMotorHBridge project, the difference is that we use an Android device to control the motor instead of two pushbuttons and a potentiometer.

The project includes the B4R program managing the Arduino UNO and the HC-05 board.

And a B4A program to communicate between the Android device and the Arduino.

The two programs HC05DCMotor.b4r and HC05DCMotor.b4a are available in the SourceCode folder.



We have a button to connect the HC-05 board.

One Togglebutton to switch ON or OFF the motor.

Another Togglebutton to change the rotation direction.

A Slider to adjust the motor speed.

We use a specialized [H-bridge](#) integrated circuit to supply the power to the DC motor, a [L293D](#) circuit. This circuit allows to change the motion direction of the motor.

This circuit needs two power supplies, one 5V supply for its internal logic circuits, pin 16, and a power supply for the motor (5 to 36V), a 9V battery in our example (pin 8).

Pins 4 and 5 are the GND pins, the two grounds (5V and 9V must be at the same level).

Pin 1 is used to manage the motor speed, a PWM signal.

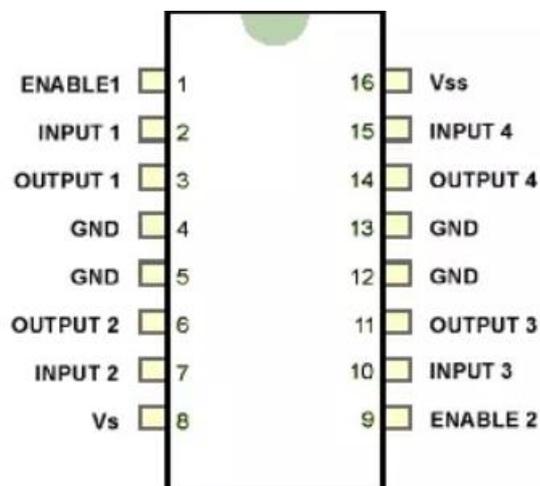
Pin 2 and pin 7 are used to change the motion direction.

Pin 2 LOW and pin 7 HIGH one direction.

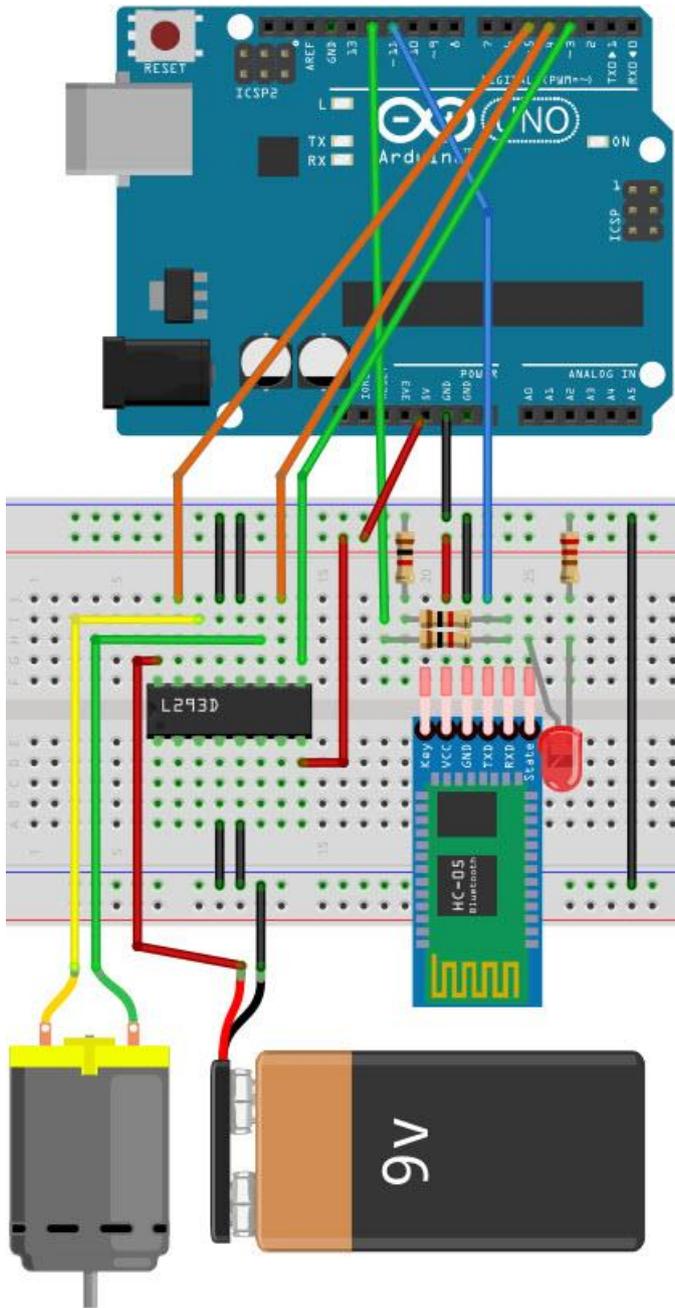
Pin 2 HIGH and pin 7 LOW the opposite direction.

Both pins LOW or both HIGH, the motor stops.

Pin 3 and pin 6 provide the power for the motor.



6.4.1 Sketch



This project is a combination of an HC-05 Bluetooth module and a DC motor.

The HC-05 sketch is the same as for the other HC-05 projects.

The DC motor sketch is almost the same as for the DCMotorHBridge project.

Material:

- 1 Arduino UNO
- 1 HC-05 board
- 1 red LED, shows the HC-05 state.
- 3 1k Ω resistors for the voltage divider.
- 1 220 Ω resistors for the LED.
- 1 DC motor [TFK-280SA-22125](https://www.aliexpress.com/item/TFK-280SA-22125/).
- 1 9V battery.
- 1 IC [L293D](https://www.aliexpress.com/item/L293D/) circuit.

6.4.2 Code

6.4.2.1 B4R Arduino

We declare all the components.

```
Sub Process_Globals
    Public Serial1 As Serial
    Private SoftwareSerial1 As SoftwareSerial
    Private astream As AsyncStreams
    Private MotorDirection1, MotorDirection2, MotorSpeed As Pin
    Private Speed As UInt
End Sub
```

We initialize the pins and the serial communication.

```
Private Sub AppStart
    Serial1.Initialize(115200)
    Log("AppStart")
    MotorDirection1.Initialize(4, MotorDirection1.MODE_OUTPUT)
    MotorDirection2.Initialize(5, MotorDirection2.MODE_OUTPUT)
    MotorSpeed.Initialize(3, MotorSpeed.MODE_OUTPUT)
    SoftwareSerial1.Initialize(9600, 11, 12) 'software serial port on pins 12 and 11
    astream.Initialize(SoftwareSerial1.Stream, "astream_NewData", Null)
End Sub
```

We use two bytes for the communication.

- The first byte is the message index.
- The second byte is the value to transmit.

```
Sub AStream_NewData (Buffer() As Byte)
    Select Buffer(0)
        Case 0 'motor On / Off
            If Buffer(1) = 0 Then
                MotorSpeed.DigitalWrite(False)
                Log("Motor OFF")
            Else
                MotorSpeed.DigitalWrite(True)
                Log("Motor ON")
            End If
        Case 1 'motor direction
            If Buffer(1) = 0 Then
                MotorDirection1.DigitalWrite(True)
                MotorDirection2.DigitalWrite(False)
                Log("Direction >>>")
            Else
                MotorDirection1.DigitalWrite(False)
                MotorDirection2.DigitalWrite(True)
                Log("Direction <<<")
            End If
        Case 2 'motor speed
            Speed = Buffer(1)
            MotorSpeed.AnalogWrite(Speed)
            Log("Speed: ", Speed)
    End Select
End Sub
```

6.4.2.2 B4A Android

Only the specific code for the motor control is explained.

The messages sent to the Arduino are arrays of two bytes.

- First byte = message index.
- Second byte = value.

We add, in Sub Globals, three variables for the message indexes for a better understanding.

Sub Globals

```
Private MsgMotorOnOff = 0 As Byte
Private MsgMotorDirection = 1 As Byte
Private MsgMotorSpeed = 2 As Byte
```

When the MotorOnOff Togglebutton is pressed, we send the message.

Byte(0) = 0, MotorOnOff message.

Byte(1) = 0 or 1, False or True.

```
Private Sub tgbMotorOnOff_CheckedChange(Checked As Boolean)
    Dim b As Byte
    If Checked = True Then
        b = 1
    Else
        b = 0
    End If
    CallSub2(Starter, "SendMessage", Array As Byte(MsgMotorOnOff, b))
End Sub
```

When the MotorDirction Togglebutton is pressed, we send the message.

Byte(0) = 1, MotorDirection message.

Byte(1) = 0 or 1, False or True.

```
Private Sub tgbMotorDirection_CheckedChange(Checked As Boolean)
    Dim b As Byte
    If Checked = True Then
        b = 1
    Else
        b = 0
    End If
    CallSub2(Starter, "SendMessage", Array As Byte(MsgMotorDirection, b))
End Sub
```

When the MotorSpinner is moved, we send the message.

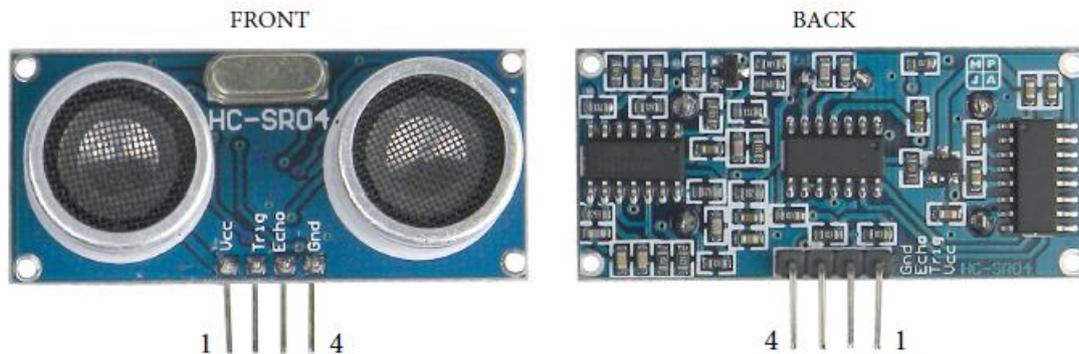
Byte(0) = 0, MotorSpeed message.

Byte(1) = speed value, between 0 and 255.

```
Sub skbMotorSpeed_ValueChanged (Value As Int, UserChanged As Boolean)
    lblMotorSpeed.Text = Value
    Dim b As Byte
    b = Value
    CallSub2(Starter, "SendMessage", Array As Byte(MsgMotorSpeed ,b))
End Sub
```

7 HC-SR04 Ultrasonic Range Sensor

The HC-SR04 is an Ultrasonic Range Sensor. It uses non-contact ultrasound sonar to measure the distance to an object – they are great for any obstacle avoiding systems on robots or rovers!



The HC-SR04 consists of an ultrasonic transmitter, an ultrasonic receiver, and a control circuit.

It offers excellent non-contact range detection from 2cm to 400 cm or 1" to 13 feet. Its operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect).

Ultrasonic Distance Measurement Principles.

The transmitter emits 8 bursts of a directional 40KHz ultrasonic wave when triggered and starts a timer. Ultrasonic pulses travel outward until they encounter an object. The object causes the wave to be reflected back towards the unit. The ultrasonic receiver would detect the reflected wave and stop the timer. The velocity of the ultrasonic burst is 340m/sec. in air. Based on the number of counts by the timer, the distance can be calculated between the object and transmitter. The TRD Measurement formula is expressed as: $D = C \times T$ which is known as the time/rate/distance measurement formula where D is the measured distance, R is the propagation velocity (Rate) in air (speed of sound) and T represents time. In this application T is divided by 2 as T is double the time value from transmitter to object back to receiver.

Features:

- Power Supply: +5V DC
- Quiescent Current: < 2mA
- Working Current: 15mA
- Effectual Angle: < 15°
- Ranging Distance: 2cm – 400 cm / 1" - 13ft
- Resolution: 0.3 cm
- Measuring Angle: 30 degrees
- Trigger Input Pulse width: 10 μ s
- Dimensions: 45mm x 20mm x 15mm

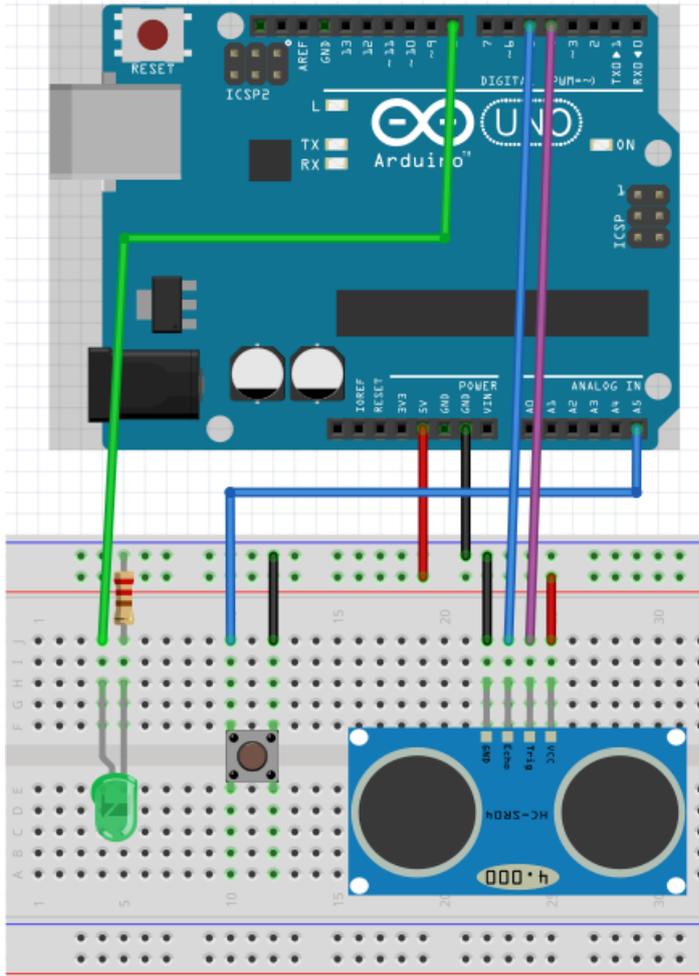
Pin assignment:

1. Vcc +5V power supply
2. Trig Trigger input pin
3. Echo Echo output pin
4. Gnd power ground

7.1 HC-SR04 Simple demo project

This project shows the basic operation of the device.

7.1.1 Sketch



We use:

- 1 Arduino UNO
- 1 HC-SR04 Ultrasonic Range Sensor
- 1 push button
- 1 LED
- 1 220 Ω resistor for the LED

We connect_

- The GND pin of the UNO to the GND line of the breadboard.
- The 5V pin of the UNO to the Vcc line of the breadboard.
- The GND pin of the HC-SR04 to the GND line of the breadboard.
- The Vcc pin of the HC-SR04 to the Vcc line of the breadboard.
- Pin D4 of the UNO to the Trig pin of the HC-SR04.
- Pin D5 of the UNO to the Echo pin of the HC-SR04.
- Pin D8 of the UNO to the LED, the other pin one via a 200 Ω resistor to GND.
- Pin A5 of the UNO to one pin of the push button, and the other pin to GND.

The push button starts or stops measuring.
The LED shows the measuring state.

7.1.2 Code

7.1.2.1 B4R Arduino

We declare all pins and variables.

8 ESP8266 / WeMos board D1 R2

Another interesting board is the ESP8266 board for IoT solutions, it's a powerful microcontroller with built-in support for wifi.

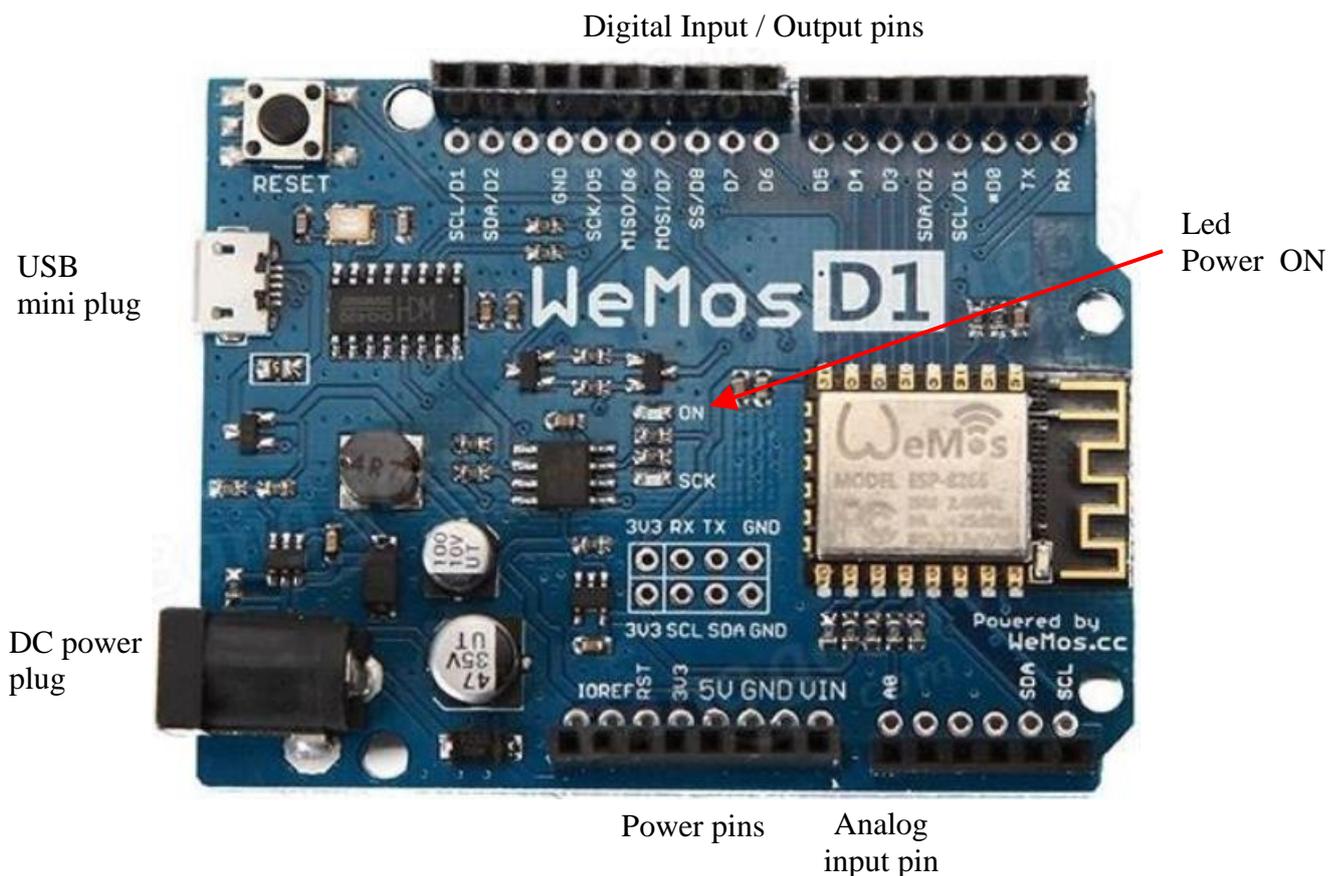
A good solution is the WeMos D1R2 board which includes a USB to serial converter.

The WeMos-D1R2 is an ESP8266-12 based WiFi enabled microprocessor unit on a Arduino-UNO footprint. That means the board looks and works (in most cases) like an UNO. Apparently, several shields, sensors and output devices that are manufactured for the Arduino platform will work on the WeMos-D1R2 with the added advantage of built-in WiFi.

There are two WeMos boards in the market.

If you are not careful you will end up with a D1 Board which is an older version.

You have to make sure you have the current version of the board WeMos-D1R2.



8.1 Difference in pin assignment Ardiono Uno > WeMos

Though the Arduino UNO and the WeMos-D1R2 are similar, there are a few differences in their pin assignment. In some situations, programs written for the UNO will need to be modified a little to the proper pin assignments of the WeMos-D1R2.



Attention: The supply voltage of the WeMos-D1R2 is 3.3 Volt !
The Arduino has 5 Volt.

Pin assignments.

The WeMos has only 11 Digital pins (RX, TX, D0 – D8) and only 1 Analog input pin. Whereas, the Arduino UNO has 16 Digital pins and 6 Analog input pins.



SCL / D1
SDA / D2
GND
GND
SCK / D5
MISO / D6
MOSI / D7
SS / D8
D7
D6
D5
D4
D3
D2
D1
D0
TX
RX

WeMos



AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1 / TX
D0 / RX

UNO



IOREF
RST
3.3 V
5 V
GND
GND
VIN
A0
SDA
SCL

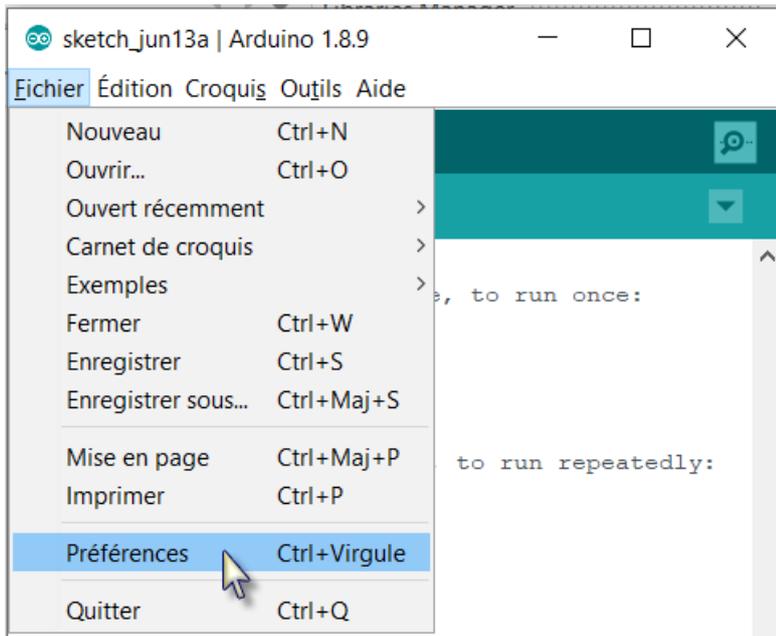
WeMos



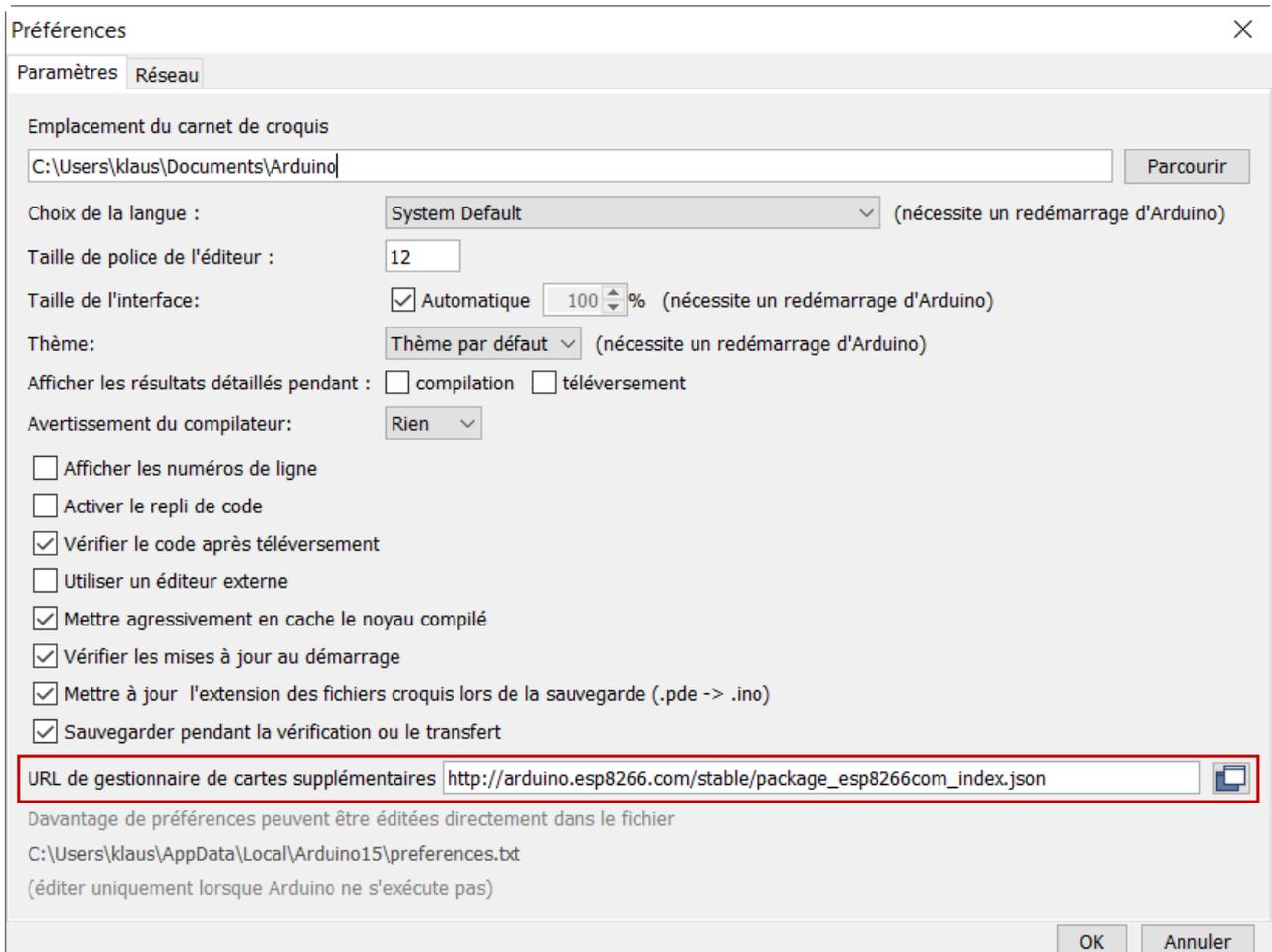
IOREF
RESET
5 V
GND
GND
VIN
A0
A1
A2
A3
A4
A5

UNO

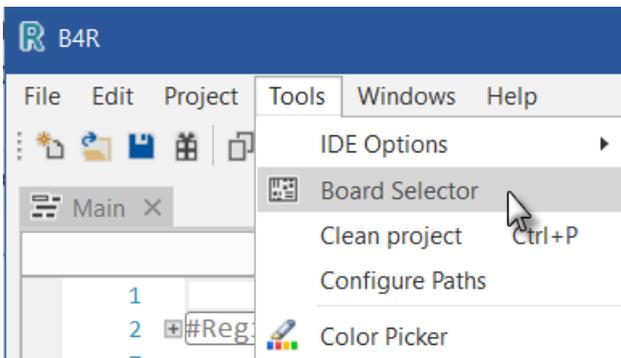
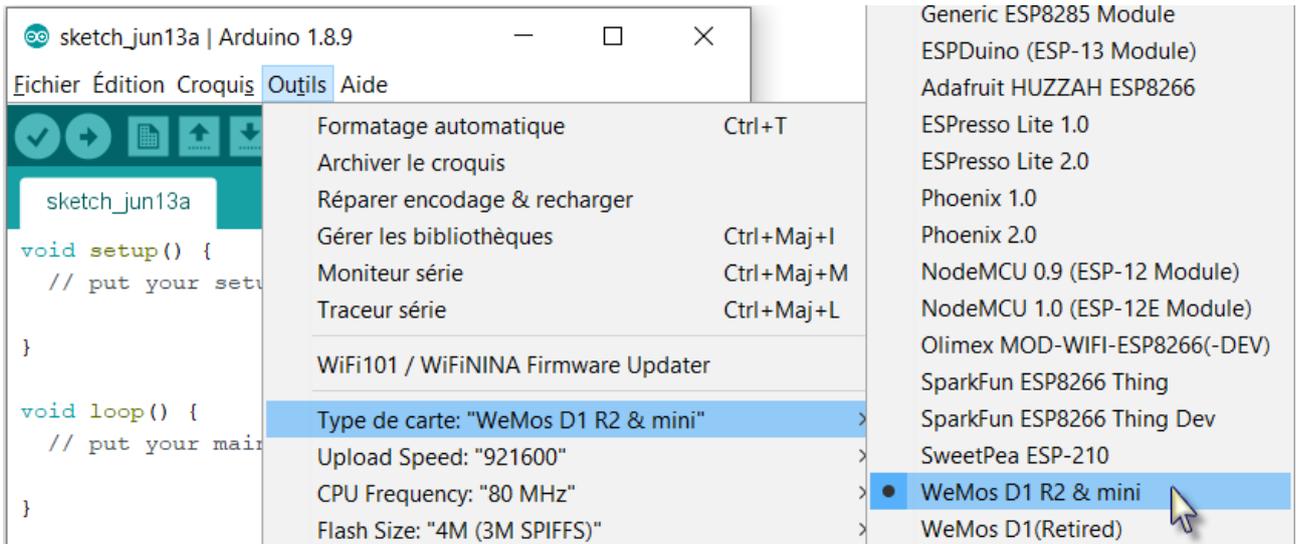
8.2 Configuration



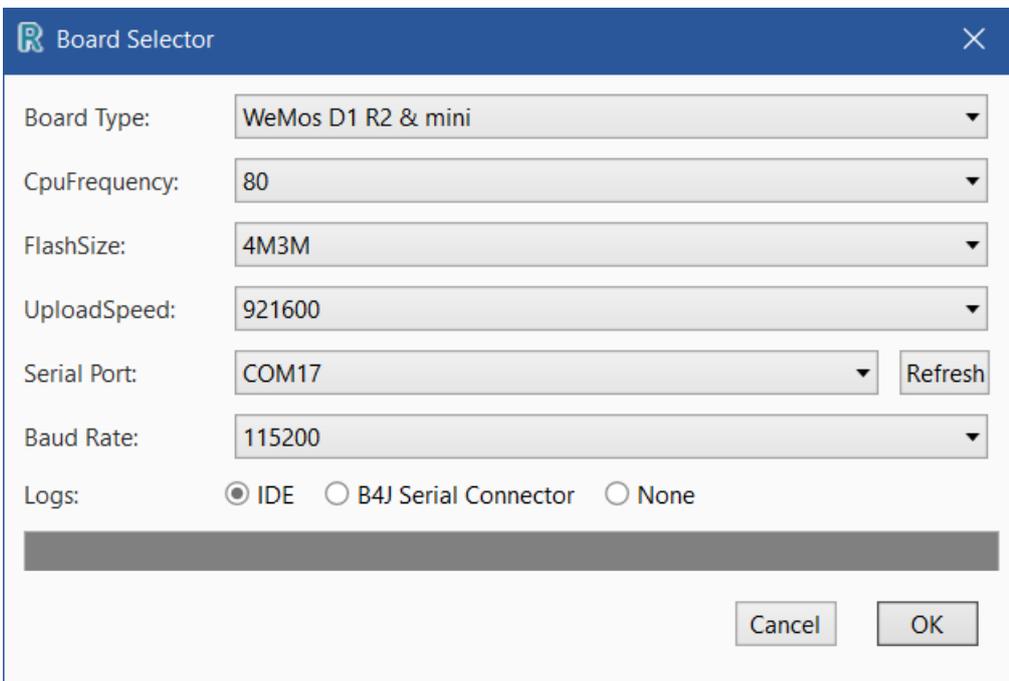
Open Arduino IDE - File - Preferences and add the following URL:
http://arduino.esp8266.com/stable/package_esp8266com_index.json



Then, in Arduino IDE > Tools - Board - Boards Manager. Search for esp and install esp8266 by ESP8266 community.



In the B4R JDE open the boards selector and select the board type (select the highest UploadSpeed):



B4R includes two ESP8266 specific libraries:

rESP8266

- ESP8266 - Currently includes a single method that restarts the board.
- D1Pins - Maps the pins of WeMos boards.

rESP8266WiFi - Similar to rEthernet library. It includes the following types:

- ESP8266WiFi - Responsible for connecting or creating the wireless network.
- WiFiSocket - Equivalent to EthernetSocket.
- WiFiServerSocket - Equivalent to EthernetServerSocket.
- WiFiUDP - Equivalent to EthernetUDP

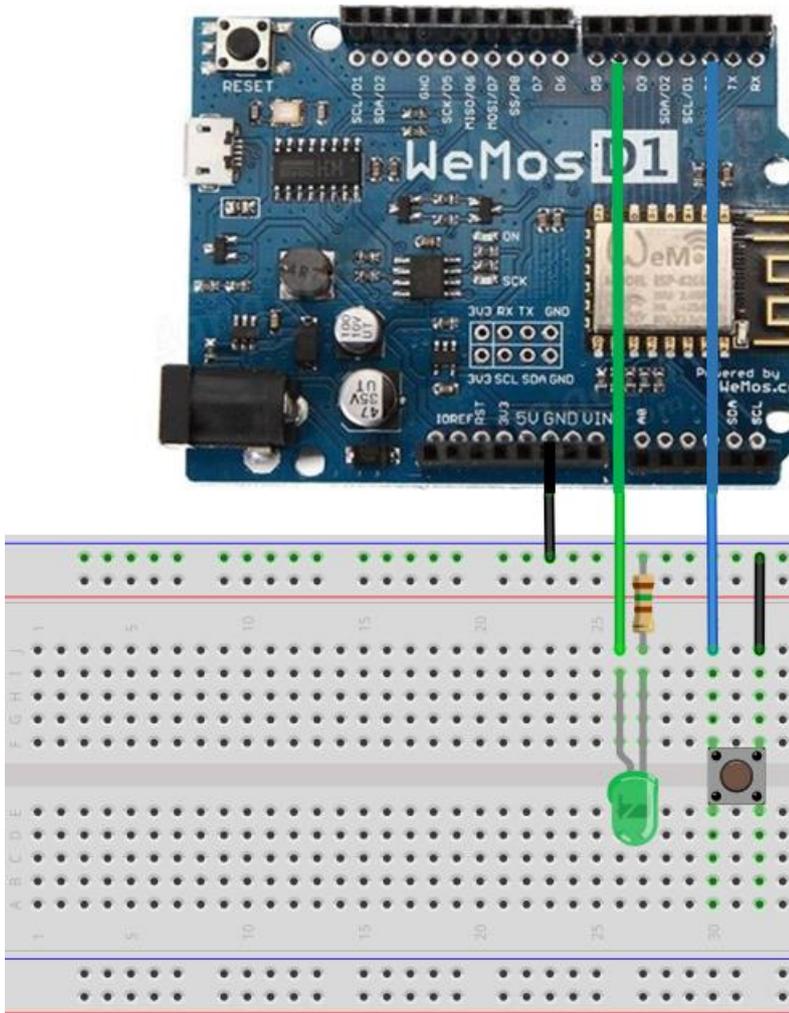
Working with ESP8266WiFi is simple and similar to working with the Ethernet shield. Example of a socket connection (depends on rESP8266WiFi and rRandomAccessFile). Note that it requires B4R v1.50+ as it uses the new [B4RSerializator](#) feature:

8.3 ESD_LEDGreen.b4r

To test the board we use a program similar to the LedGreen,b4r project.

A pushbutton to switch ON or OFF a green LED.

8.3.1 Sketch



Material:

- 1 pushbutton switch
- 1 green LED
- 1 150 Ω resistor

We connect one GND pin of the WeMos to the GND line of the breadboard.

We add a pushbutton on the breadboard. Connect one pin to the breadboard GND line.

The other pin to digital pin D0 on the WeMos.

Then we

- Add a green Led on the breadboard.
- Connect the cathode (-) via a 150 Ω resistor to the ground GND line of the breadboard.
- Connect the anode (+) to digital pin D4.

For the LED we use a resistor with a value of 150 Ω instead of 220 Ω because the voltage of the WeMos is 3.3 V instead of 5 V for the Arduino ONE.

8.3.2 Code

This project needs the rESP266 library.

```
Sub Process_Globals
  Public Serial1 As Serial
  Private pinButton As Pin           'pin for the button
  Private pinLEDGreen As Pin       'pin for the green Led
  Private LightOn = False As Boolean

  Private ESPins As D1Pins
End Sub
```

We reuse the same declarations as in the Arduino LEDGreen project, but we need to add a declaration, `Private ESPins As D1Pins`, for the pins of the WeMos.

```
Private Sub AppStart
  Serial1.Initialize(115200)

  'Using the internal pull up resistor to prevent the pin from floating.
  pinButton.Initialize(ESPins.D4, pinButton.MODE_INPUT_PULLUP)
  pinButton.AddListener("pinButton_StateChanged")

  pinLEDGreen.Initialize(ESPins.D5, pinLEDGreen.MODE_OUTPUT)
End Sub
```

The initialization of the pins is different, we use the D1Pins object instead of the Pin object.

```
Private Sub pinButton_StateChanged (State As Boolean)
  If State = False Then 'remember, False means button pressed.
    LightOn = Not(LightOn)
    pinLEDGreen.DigitalWrite(LightOn)
  End If
End Sub
```

This routine the same as in the Arduino LEDGreen project.

8.4 WiFi Remote Configuration

Empty.

9 FAQ

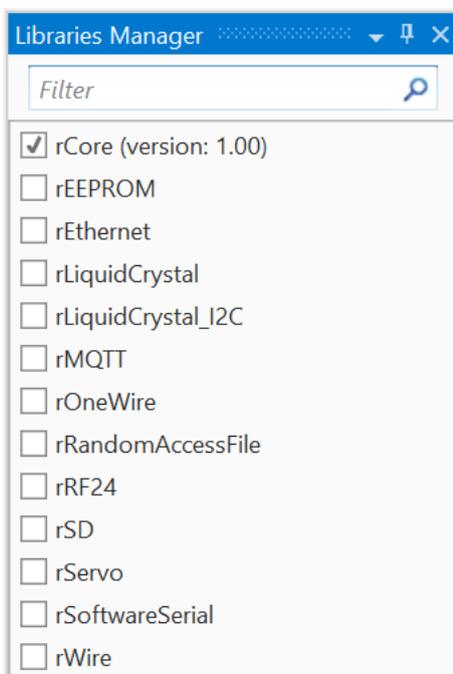
Some of the chapters below have been picked up from the forum.

9.1 "Please save project first" message

When I try to compile or open the Designer, I see a message saying: "Please save source code first." A new project doesn't have a containing folder until it is first saved. Save your project and this error will go away.

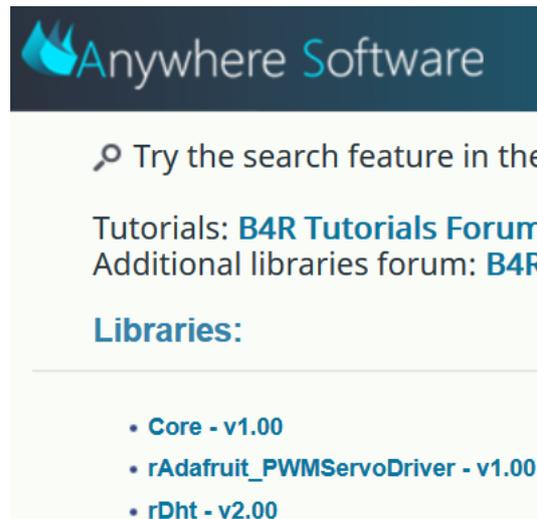
9.2 "Are you missing a library reference" message

Compiler says: "Are you missing a library reference?".



Go to the Libraries tab in the right pane and check the required libraries.

If you do not know which library a specific object type belongs to, you can go to the [documentation](#) page.



Types:

- [Adafruit_PWMServoDriver](#)
- [AsyncStreams](#)
- [Bit](#)
- [ByteConverter](#)
- [dht](#)
- [EEPROM](#)
- [Ethernet](#)

At the bottom of this page there is a long list with all the object types.

Pressing on any type will take you to the right library. Note that the trial version doesn't support libraries. Only the full version.

9.3 How loading / updating a library

See the Libraries chapter in the [B4X B4X Language](#) booklet.

A list of the official and additional libraries with links to the relevant forum threads is shown in the [B4R Documentation](#) page.

To load or update a library follow the steps below:

- Download the library zip file somewhere.
- Unzip it.
- Copy the xxx.xml file to the
 - B4R Library folder for a standard B4R library
 - Additional libraries folder for an additional library.
- Right click in the libraries list in the Tab LibrariesManager and click on  and select the library.

9.4 Split a long line into two or more lines

To split a long line into two or more lines put an underscore character, separated by a blank character, at the end of the line.

```
pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP) 'Using t
```

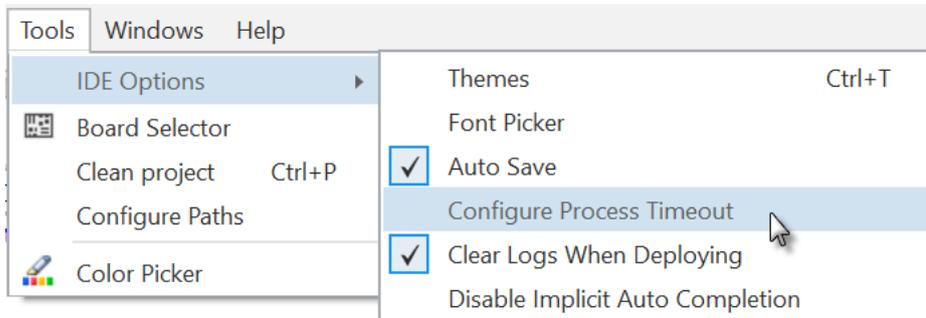
Becomes:

```
pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP) _  
'Using the internal pull up resistor to prevent the pin from floating.
```

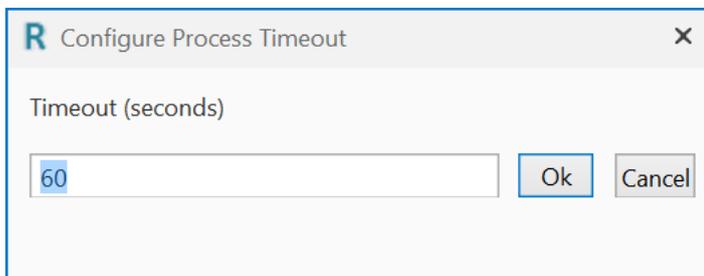
9.5 "Process has timeout" message

If you often get this message "Process has timeout" you can change its value:

- In the IDE menu Tools / IDE Options click on Configure Process Timeout.



- And change the value:



9.6 How to pass an Array to a Sub

It is possible to pass Arrays, also multidimensional Arrays, to a sub.

Code example.

```
Private one(1), two(1,2), three(1,2,3) As String

Sub Test(a() As String, b(,) As String, c(,) As String) As String(,)
    ...
End Sub
'
'
    Test(one, two, three)
```

You need to specify the rank (number of dimensions) in the Sub definition with ','. If you want the Sub to return an array you must also specify it.

```
Sub Test(a() As String, b(,) As String, c(,) As String) As String
Returns a single string.
```

```
Sub Test(a() As String, b(,) As String, c(,) As String) As String()
Returns a one rank string array.
```

```
Sub Test(a() As String, b(,) As String, c(,) As String) As String(,)
Returns a two rank string array.
```

9.7 Select True / Case trick

The question: It would be nice to be able to use Select Case using the 'greater than' and 'less than' operators <>. It makes for cleaner code than 'if' 'else' and 'end if' etc.

This trick does it:

```
i = 10
Select True
Case (i < 9)
    Log("False")
Case (i = 10)
    Log("True")
End Select
```

10 Glossary

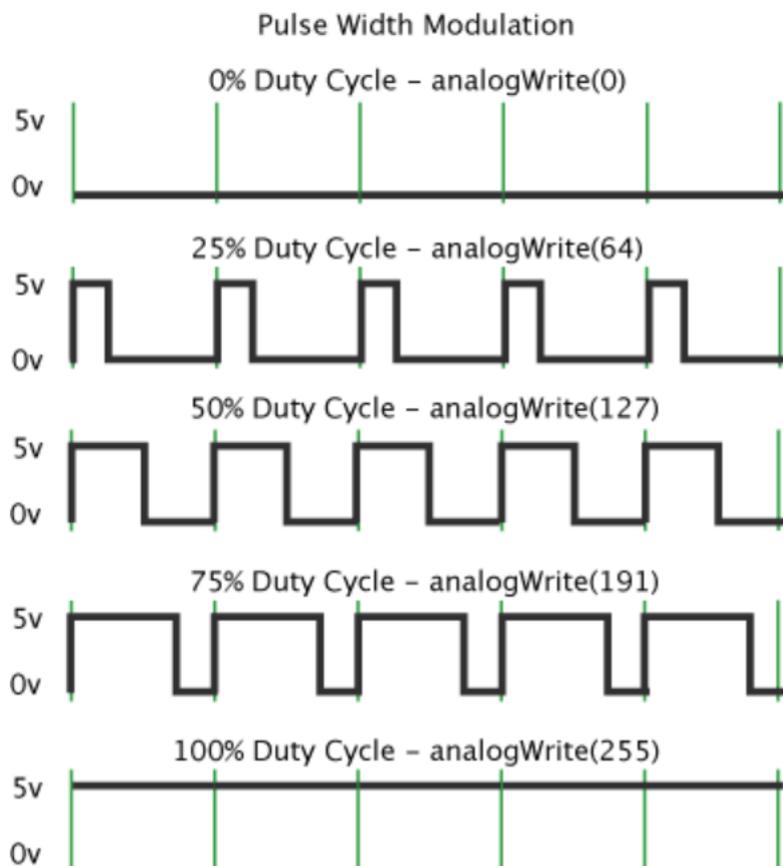
10.1 Electricity basics

[Electricity Basics](#). Link to the ITP Physical Computing site.

10.2 PWM Pulse Width Modulation

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to [analogWrite\(\)](#) is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example.



Source [Android Site Tutorials](#).