



App Inventor 2 Classic Bluetooth Extension

DESCRIPTION

This document describes the use of an App Inventor 2 (AI2) extension to be able to communicate via Classic Bluetooth an App developed with AI2 with any device. Associated with this extension, an Arduino code is included to be able to encode and decode the Bluetooth telegrams (see Protocol section).

The extension includes AI2 components to access different types of generic electronics such as digital, analog, and PWM inputs and outputs, but also for fairly common Arduino devices such as the sound buzzer, temperature and humidity sensors, infrared, etc... each These devices have an AI2 component with specific functionalities.

In addition, there are also components for access to variables, not necessarily associated with a certain type of electronics, on which we can share information between the App and the device. These variables can be of the type boolean (bool), integer (int), decimal (float) and text strings (String).

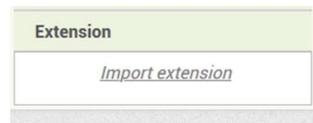
DOWNLOAD

The latest version of the extension can be downloaded from:

<https://roboticafacil.es/facilino/ai2/es.roboticafacil.facilino.runtime.bluetooth.aix>

IMPORTING EXTENSION IN AI2

Once downloaded, you can import the extension to your project from the designer view:



DEVELOPER

Please, contribute to this extension at:

https://github.com/roboticafacil/facilino_ai2



Example 1: Controlling digital output and servos

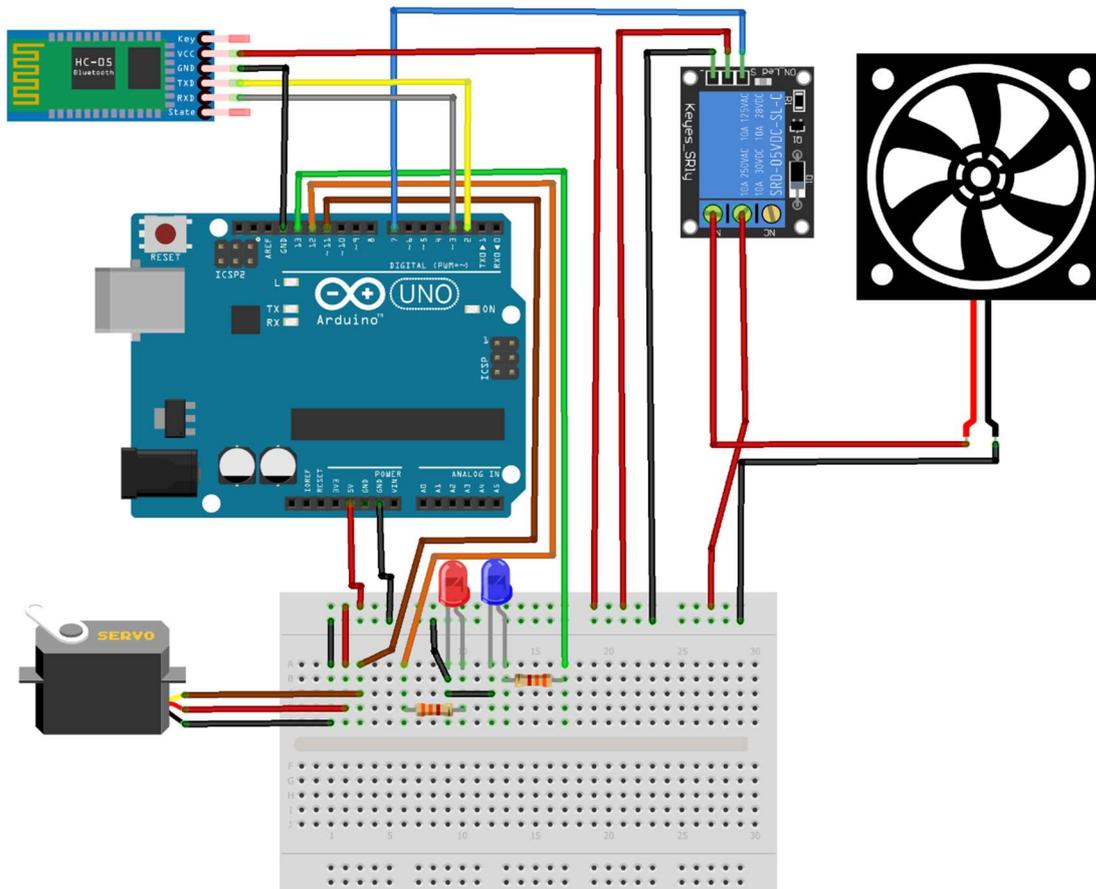
Download

https://roboticafacil.es/facilino/ai2/demos/Uno_LEDs_Servo_Relay.zip

Connection diagram

The following example uses the connection diagram as shown. The circuit includes an Arduino Uno, two LEDs, a servo, a relay and a Bluetooth module HC-05 connected as follow:

- Red LED pin 12.
- Blue LED pin 13.
- Relay module pin 7.
- Servo pin 11.
- Bluetooth module connected to pin D2 (TX) and pin D3 (RX).



fritzing

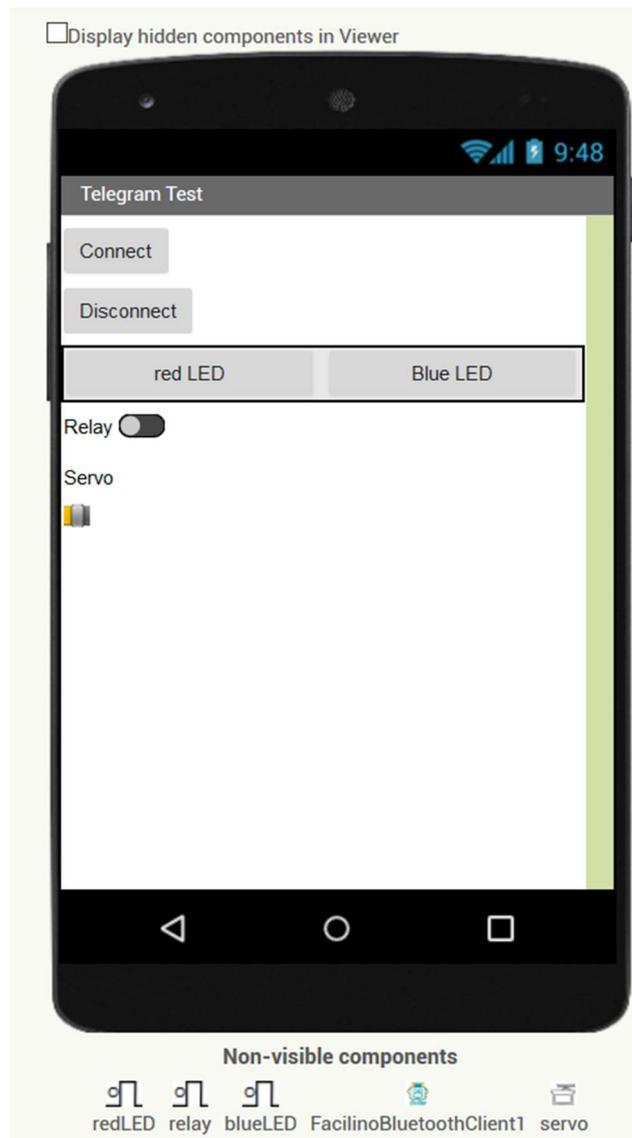
App Inventor

Designer instructions

- Create a **ListPicker** component and rename it as 'connect'.
- Create a **Button** component and rename it as 'disconnect'.
- Create two **Buttons** components inside a **HorizontalArrangement** and rename them as 'butRed' and 'butBlue'.
- Create **Switch** component and rename it as 'relaySwitch'.
- Create a **Slider** component and rename it as 'servoSlider'.
- Import the **FacilinoBluetooth AI2** extension ([es.roboticafacil.facilino.runtime.bluetooth.ai2](https://www.appinventor.ai/es.roboticafacil.facilino.runtime.bluetooth.ai2)).
- Create a **FacilinoBluetoothClient** component (default name 'FacilinoBluetoothClient1').



- Create a **DigitalWriteBluetooth** component and rename it as *'redLED'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 12.
- Create a **DigitalWriteBluetooth** component and rename it as *'blueLED'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 13.
- Create a **DigitalWriteBluetooth** component and rename it as *'relay'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 7.
- Create a **ServoBluetooth** component and rename it as *'servo'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 11.



Blocks

Please, refer to the [REFERENCE MANUAL](#) of **FacilinoBluetooth** extension for further information on each, method, property or event.

Connection and disconnection use the *'Connect'* and *'Disconnect'* methods of the **FacilinoBluetoothClient** component. To list the MAC address in the **ListPicker** component, use the *'AddressesAndNames'* property. To check if Bluetooth is *'Enabled'* and *'Available'*, use the corresponding properties.



```
when connect .BeforePicking
do
  if
    FacilinoBluetoothClient1 .Enabled and FacilinoBluetoothClient1 .Available
  then
    set connect .Elements to FacilinoBluetoothClient1 .AddressesAndNames

when connect .AfterPicking
do
  evaluate but ignore result
  call FacilinoBluetoothClient1 .Connect
  address connect .Selection

when disconnect .Click
do
  call FacilinoBluetoothClient1 .Disconnect
```

To set digital outputs, we can use either the 'Set' method or the 'Toggle' method. To set the position of the servo, we can use the 'Move' method.

```
initialize global red to false
initialize global blue to false

when butRed .Click
do
  set global red to not get global red
  call redLED .Set
  value get global red

when butBlue .Click
do
  set global blue to not get global blue
  call blueLED .Set
  value get global blue

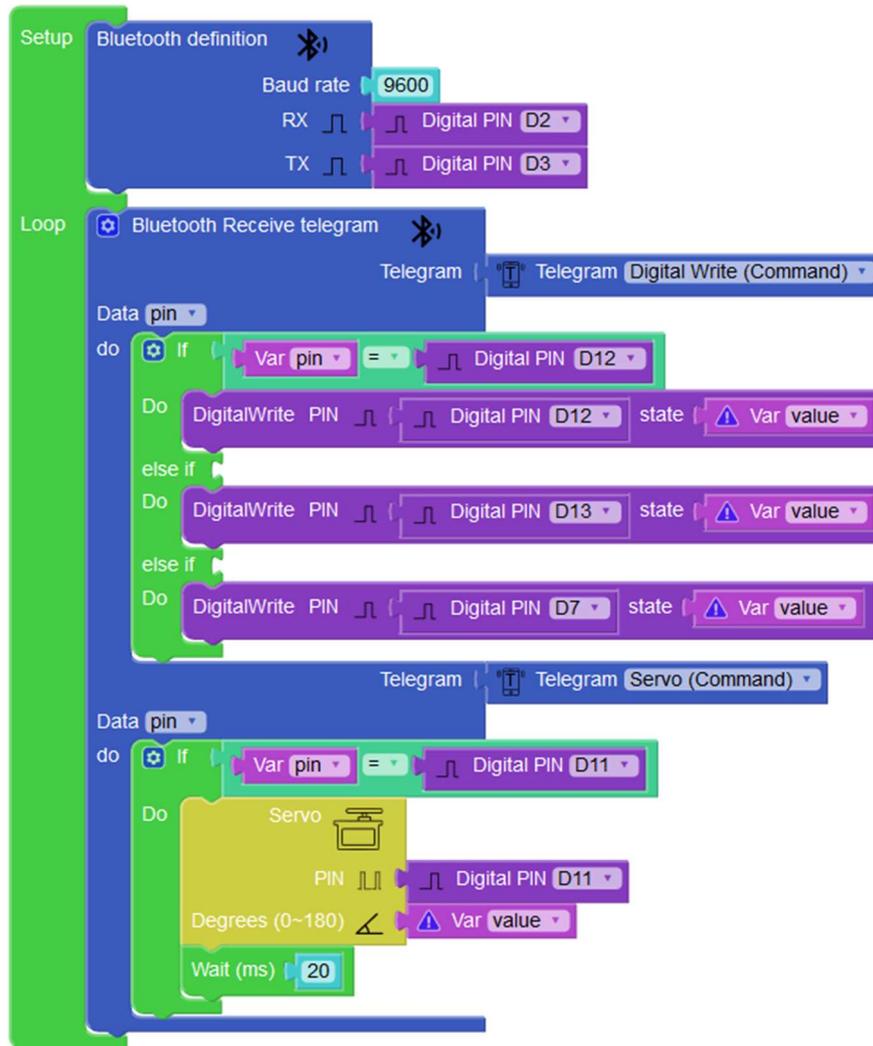
when relaySwitch .Changed
do
  call relay .Toggle

when servoSlider .PositionChanged
thumbPosition
do
  call servo .Move
  angle get thumbPosition
```

Facilino Code

The *Bluetooth definition* instruction uses pins D2 and D3, to establish the software serial connection (please, consider that RX and TX between the Bluetooth module and Arduino are crossed).

The *Bluetooth Receive telegram* instruction decodes any received telegram. In this case, we are decoding *Digital Write* telegram and *Servo Telegram*. We must check the pin number before using the *DigitalWrite* or *Servo* instructions.



Arduino Code

The following code decodes two types of telegrams, *Digital Write* and *Servo*, with commands 0x02 and 0x10, respectively. See [PROTOCOL MANUAL](#) for further details. First, we initialise the Bluetooth module with 9600bauds, based on the **SoftwareSerial** library and set the pin modes to the corresponding mode as well as the servo attach. Then, in the loop function, we implement the actual telegram decoding, so in line 54 checks the type of telegram received, obtaining meaningful information form the telegram data such as pin and value. Depending on the receive telegram, the code between lines 54 and 71 performs the corresponding action.

```
1 #include <SoftwareSerial.h>
2 #include <Servo.h>
3
4 #define BT_TX_PIN 2 //TX pin of bluetooth module
5 #define BT_RX_PIN 3 //RX pin of bluetooth module
6 #define RED_LED_PIN 12
7 #define BLUE_LED_PIN 13
8 #define RELAY_PIN 7
9 #define SERVO_PIN 11
10
11 SoftwareSerial _bt_device(BT_TX_PIN,BT_RX_PIN);
12
13 int _bt_pos=0;
14 unsigned char _bt_cmd=0;
15 int _bt_length=0;
16 unsigned char _bt_data[255];
17
18 #define CMD_DIGITAL_WRITE 0x02
```



```
19 #define CMD_SERVO 0x10
20
21 Servo _servo;
22
23 void setup()
24 {
25     _bt_device.begin(9600);
26     _bt_device.flush();
27     pinMode(RED_LED_PIN, OUTPUT);
28     pinMode(BLUE_LED_PIN, OUTPUT);
29     pinMode(RELAY_PIN, OUTPUT);
30     _servo.attach(SERVO_PIN);
31 }
32
33 void loop()
34 {
35     if (_bt_device.available()>0)
36     {
37         unsigned char c;
38         _bt_device.readBytes(&c,1);
39         if ((c=='@')&&(_bt_pos==0))
40             _bt_pos++;
41         else if (_bt_pos==1){
42             _bt_pos++;
43             _bt_cmd=c;
44         }
45         else if (_bt_pos==2){
46             _bt_pos++;
47             _bt_length=c;
48         }
49         else if ((_bt_pos>=3)&&(_bt_pos<=( _bt_length+2))){
50             _bt_data[_bt_pos-3]=c;
51             _bt_pos++;
52         }
53         else if ((_bt_pos==( _bt_length+3))&&(c=='*')){
54             if (_bt_cmd==CMD_DIGITAL_WRITE){
55                 int pin = _bt_data[0];
56                 boolean value = _bt_data[1]==1? HIGH: LOW;
57                 if (pin==RED_LED_PIN)
58                     digitalWrite(RED_LED_PIN,value);
59                 else if (pin==BLUE_LED_PIN)
60                     digitalWrite(BLUE_LED_PIN,value);
61                 else if (pin==RELAY_PIN)
62                     digitalWrite(RELAY_PIN,value);
63             }
64             if (_bt_cmd==CMD_SERVO){
65                 int pin = _bt_data[0];
66                 byte value = _bt_data[1];
67                 if (pin==SERVO_PIN) {
68                     _servo.write(value);
69                     delay(20);
70                 }
71             }
72             _bt_pos=0;
73             _bt_length=0;
74         }
75         else{
76             _bt_pos=0;
77             _bt_length=0;
78         }
79     }
}
```



Example 2: Multisensor Shield with Arduino Uno

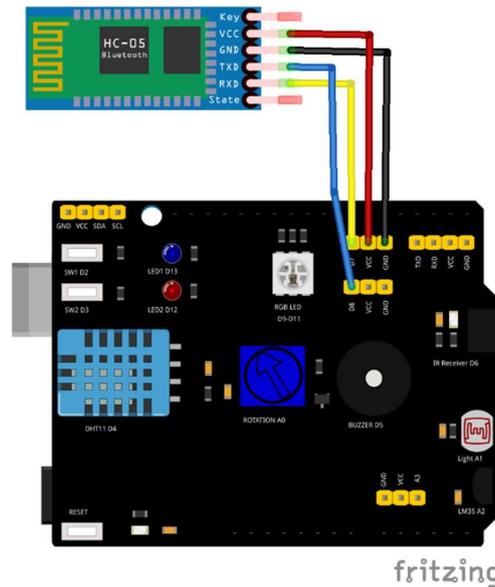
Download

https://roboticafacil.es/facilino/ai2/demos/Multisensor_ArduinoUno.zip

Connection diagram

The following example uses the connection diagram as shown. The circuit includes an Arduino Uno a multisensor shield and a Bluetooth module HC-05 connected as follow:

- Multisensor shield (<https://roboticafacil.es/en/prod/multisensor-shield/>)
- Red LED connected to D12.
- Blue LED connected to D13.
- SW1, push-button, connected to D2.
- SW2, push-button, connected to D3.
- DHT11, temperature and humidity sensor, connected to D4.
- Buzzer connected to D5.
- RGB LED connected to pins D9-D11 and D10.
- Potentiometer connected to A0.
- LDR connected to A1.
- Bluetooth module connected to pin D2 (TX) and pin D3 (RX).



App Inventor

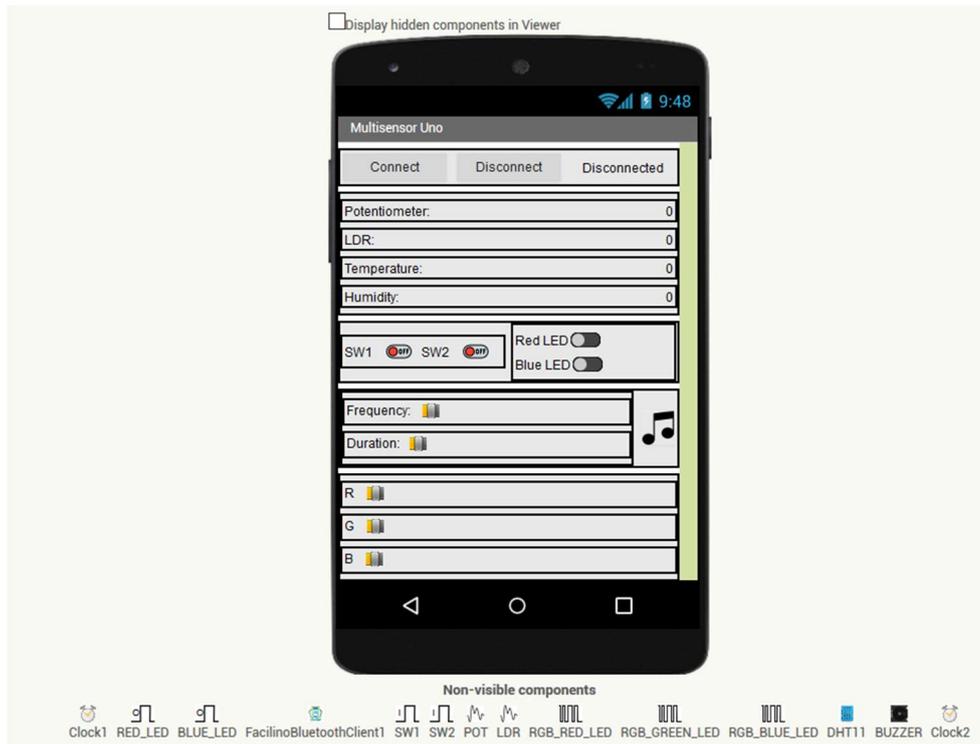
Designer instructions

- Create a **ListPicker** component and rename it as *'connect'*, create a **Button** component and rename it as *'disconnect'* and create a **Label** component and rename it as *'connStatus'*. All these components must be horizontally arranged with a **HorizontalArrangement** component. Make sure that the **HorizontalArrangement**, **ListPicker** and **Button** components fill the parent width (in the *'Width'* property) and modify the *'Text'* property as shown in the image in the **ListPicker** and **Button** components.
- Create two **Label** components, one with a fixed text and the other one with a number indicating the magnitude of the variable read (initially set this value to 0). These two labels will be inside a **HorizontalArrangement** component. Repeat this procedure to create labels to measure the potentiometer, the LDR, the temperature and humidity. Rename the **Label** components correspondingly (the ones used to show the magnitude), i.e.: *'PotVal'*, *'LDRVal'*, *'tempVal'* and *'humidVal'*. Make sure that the numeric **Label** and **HorizontalArrangement** components fill the parent width (in the *'Width'* property). Modify the *'Text'* property of static **Label** components as shown in the image.
- Inside a **HorizontalArrangement** component create nested **HorizontalArrangement** and **VerticalArrangement** components. Inside the nested **HorizontalArrangement** component create a **Label** component, an **Image**



component a second **Label** component and a second **Image** component. Inside the nested **VerticalArrangement** component, create two **Switch** components. Rename the **Image** components with *'SW1_Img'* and *'SW2_Img'* and the **Switch** components as *'Switch1'* and *'Switch2'*. Obtain the corresponding layout, as shown in the image, by modifying the *'Width'* property and also modify the *'Text'* property conveniently.

- Create a **HorizontalArrangement** component and a nested **VerticalArrangement** component and a **Button** component. Inside the **VerticalArrangement** component create two **HorizontalArrangement** components, each with a **Label** with a fixed text and a **Slider** component. Rename the **Button** component as *'playSound'* and the **Slider** components as *'freq_slider'* and *'duration_slider'*. Obtain the corresponding layout, as shown in the image, by modifying the *'Width'* property and also modify the *'Text'* property conveniently and even for the **Button** component, select a music note icon (obtained from i.e.: flaticon.com, select the one you prefer).
- Create three **HorizontalArrangement** components, each with a **Label** with a fixed text and a **Slider** component. Rename the Slider components as *'Red_slider'*, *'Green_slider'* and *'Blue_slider'*. Obtain the corresponding layout, as shown in the image, by modifying the *'Width'* property and also modify the *'Text'* property conveniently.
- Create two **Clock** components and set the *'TimerInterval'* property to 100 (milliseconds) for *'Clock1'* and 2000 for *'Clock2'*.
- Import the **FacilinoBluetooth** AI2 extension (es.roboticafacil.com/facilino/runtime/bluetooth.aix).
- Create a **FacilinoBluetoothClient** component (default name *'FacilinoBluetoothClient1'*).
- Create two **DigitalWriteBluetooth** components and rename them as *'RED_LED'* and *'BLUE_LED'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 12 and 13, respectively.
- Create a **DigitalReadBluetooth** components and rename them as *'SW1'* and *'SW2'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to numbers 2 and 3, respectively.
- Create two **AnalogReadBluetooth** components and rename them as *'POT'* and *'LDR'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to numbers 14 and 15 (in Arduino Uno A0 is number 14, while A1 is number 15).
- Create three **AnalogWriteBluetooth** components and rename them as *'RGB_RED_LED'*, *'RGB_GREEN_LED'* and *'RGB_BLUE_LED'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to numbers 9, 11, and 10, respectively.
- Create a **DHTBluetooth** component and rename it as *'DHT11'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 4.
- Create a **BuzzerBluetooth** component and rename it as *'BUZZER'* and assign the property *'FacilinoBluetoothClient'* to point the **FacilinoBluetoothClient** component. Also modify the *'Pin'* property to number 4.



Blocks

Please, refer to the [REFERENCE MANUAL](#) of **FacilinoBluetooth** extension for further information on each, method, property or event.

Connection and disconnection use the 'Connect' and 'Disconnect' methods of the **FacilinoBluetoothClient** component. To list the MAC address in the **ListPicker** component, use the 'AddressesAndNames' property. To check if Bluetooth is 'Enabled' and 'Available', use the corresponding properties.

```
when connect BeforePicking
do
  if FacilinoBluetoothClient1 Available
  then set connect Elements to FacilinoBluetoothClient1 AddressesAndNames

when connect AfterPicking
do
  if call FacilinoBluetoothClient1 .Connect
  address connect Selection
  then set connStatus Text to "Connected"

when disconnect Click
do
  set connStatus Text to "Disconnected"
  call FacilinoBluetoothClient1 .Disconnect
```

To read from all digital and analog signals, such as the potentiometer, the LDR and the two push-buttons, we can use the 'Request' method, included on each of the corresponding components. This can be done inside the 'Timer' event of the 'Clock1' component. DHT sensor can be requested inside the 'Timer' event of 'Clock2' component. The reason because DHT Request is implemented inside Clock2 is because DHT sensor is a 'slow' sensor and requires less frequently measurements.



```

when Clock1 .Timer
do
  if FacilinoBluetoothClient1 .IsConnected
  then
    set connStatus .Text to "Connected"
    call POT .Request
    call LDR .Request
    call SW1 .Request
    call SW2 .Request
  else
    set connStatus .Text to "Disconnected"

```

```

when Clock2 .Timer
do
  if FacilinoBluetoothClient1 .IsConnected
  then
    set connStatus .Text to "Connected"
    call DHT11 .Request
  else
    set connStatus .Text to "Disconnected"

```

Upon each request, we need to handle the 'Received' events, if received.

```

when POT .Received
value
do
  set PotVal .Text to get value

```

```

when SW1 .Changed
value
do
  if get value
  then
    set SW1_img .Picture to "switch-off_24x24.png"
  else
    set SW1_img .Picture to "switch-on_24x24.png"

```

```

when DHT11 .Received
temperature humidity
do
  set tempVal .Text to get temperature
  set humidVal .Text to get humidity

```

```

when SW2 .Changed
value
do
  if get value
  then
    set SW2_img .Picture to "switch-off_24x24.png"
  else
    set SW2_img .Picture to "switch-on_24x24.png"

```

```

when LDR .Received
value
do
  set LDRVal .Text to get value

```

Finally, let's handle the digital outputs of the board, such as the red and blue LEDs, the RGB LEDs by calling the 'Set' method and the Buzzer component which can produce tones using the 'Tone' method:

```

when Switch1 .Changed
do
  if
  then
    set connStatus .Text to "Connected"
    call RED_LED .Set
    value Switch1 .On
  else
    set connStatus .Text to "Disconnected"

```

```

when Switch2 .Changed
do
  if
  then
    set connStatus .Text to "Connected"
    call BLUE_LED .Set
    value Switch2 .On
  else
    set connStatus .Text to "Disconnected"

```

```

when Red_slider .PositionChanged
thumbPosition
do
  if
  then
    set connStatus .Text to "Connected"
    call RGB_RED_LED .Set
    value Red_slider .ThumbPosition
    call RGB_GREEN_LED .Set
    value Green_slider .ThumbPosition
    call RGB_BLUE_LED .Set
    value Blue_slider .ThumbPosition
  else
    set connStatus .Text to "Disconnected"

```

```

when Blue_slider .PositionChanged
thumbPosition
do
  if
  then
    set connStatus .Text to "Connected"
    call RGB_RED_LED .Set
    value Red_slider .ThumbPosition
    call RGB_GREEN_LED .Set
    value Green_slider .ThumbPosition
    call RGB_BLUE_LED .Set
    value Blue_slider .ThumbPosition
  else
    set connStatus .Text to "Disconnected"

```

```

when Green_slider .PositionChanged
thumbPosition
do
  if
  then
    set connStatus .Text to "Connected"
    call RGB_RED_LED .Set
    value Red_slider .ThumbPosition
    call RGB_GREEN_LED .Set
    value Green_slider .ThumbPosition
    call RGB_BLUE_LED .Set
    value Blue_slider .ThumbPosition
  else
    set connStatus .Text to "Disconnected"

```

```

when playSound .Click
do
  if
  then
    set connStatus .Text to "Connected"
    call BUZZER .Tone
    frequency freq_slider .ThumbPosition
    duration duration_slider .ThumbPosition
  else
    set connStatus .Text to "Disconnected"

```



Facilino Code

The *Bluetooth definition* instruction uses pins D7 and D8, to establish the software serial connection (please, consider that RX and TX between the Bluetooth module and Arduino are crossed).

The *Bluetooth Receive telegram* instruction decodes any received telegram. In this case, we are decoding *Digital Read*, *Digital Write*, *Analog Read*, *Analog Write*, *Buzzer Tone* and *DHT* telegrams. We must check the pin number before using the corresponding instructions on each telegram.

```
Setup
  Bluetooth definition
    Baud rate 9600
    RX Digital PIN D7
    TX Digital PIN D8

Loop
  Bluetooth Receive telegram
    Telegram Digital Read (Request)
    Data pin
    do
      If Var pin = Digital PIN D2
      Do Bluetooth Send telegram
        Telegram Digital Read (Response)
        PIN Digital PIN D2
        Data (1 Byte) To number Byte DigitalRead PIN Digital PIN D2
      else if Var pin = Digital PIN D3
      Do Bluetooth Send telegram
        Telegram Digital Read (Response)
        PIN Digital PIN D3
        Data (1 Byte) To number Byte DigitalRead PIN Digital PIN D3
    Telegram Digital Write (Command)
    Data pin
    do
      If Var pin = Digital PIN D12
      Do DigitalWrite PIN Digital PIN D12 state Var value
      else if Var pin = Digital PIN D13
      Do DigitalWrite PIN Digital PIN D13 state Var value
    Telegram Analog Read (Request)
    Data pin
    do
      If Var pin = Analog PIN A0
      Do Bluetooth Send telegram
        Telegram Analog Read (Response)
        PIN Analog PIN A0
        Data (2 Bytes) AnalogRead PIN Analog PIN A0
      else if Var pin = Analog PIN A1
      Do Bluetooth Send telegram
        Telegram Analog Read (Response)
        PIN Analog PIN A1
        Data (2 Bytes) AnalogRead PIN Analog PIN A1
```



The code is structured as follows:

- Telegram Analog Write (Command):** A loop that checks the value of a variable 'pin'. If it is 9, it writes a PWM signal to pin D9. If it is 10, it writes to D10. If it is 11, it writes to D11.
- Telegram DHT (Request):** A loop that checks if a digital pin (D4) is active. If so, it declares a 4-byte array 'data', reads temperature and humidity from a DHT11 sensor, shifts the temperature value right by 8 bits, and stores the results in the array. It then sends this data via Bluetooth.
- Telegram Buzzer Tone (Command):** A loop that checks if a digital pin (D5) is active. If so, it triggers an advanced buzzer with a generic tone, frequency, and duration.



Arduino Code

The following code decodes two types of telegrams, *Digital Read*, *Digital Write*, *Analog Read*, *Analog Write*, *DHT*, and *Buzzer Tone* with commands 0x00, 0x02, 0x03, 0x05, 0x22 and 0x20, respectively. See [PROTOCOL MANUAL](#) for further details. First, we initialise the Bluetooth module with 9600bauds, based on the **SoftwareSerial** library and set the pin modes to the corresponding mode as well as the servo attach. Then, in the loop function, we implement the actual telegram decoding, so in line 71 checks the type of telegram received, obtaining meaningful information from the telegram data such as pin and value. Depending on the receive telegram, the code between lines 71 and 156 performs the corresponding action.

```
1  #include <SoftwareSerial.h>
2  #include <DHT.h>
3
4  #define BT_TX_PIN 7 //TX pin of bluetooth module
5  #define BT_RX_PIN 8 //RX pin of bluetooth module
6  #define RED_LED_PIN 12
7  #define BLUE_LED_PIN 13
8  #define SW1_PIN 2
9  #define SW2_PIN 3
10 #define DHT11_PIN 4
11 #define BUZZER_PIN 5
12 #define RGB_RED_PIN 9
13 #define RGB_GREEN_PIN 11
14 #define RGB_BLUE_PIN 10
15 #define POT_PIN A0
16 #define LDR_PIN A1
17
18 SoftwareSerial _bt_device(BT_TX_PIN,BT_RX_PIN);
19 DHT sensorDHT11(DHT11_PIN,DHT11);
20
21 int _bt_pos=0;
22 unsigned char _bt_cmd=0;
23 int _bt_length=0;
24 unsigned char _bt_data[255];
25
26 #define CMD_DIGITAL_READ_REQ 0x00
27 #define CMD_DIGITAL_READ_RESP 0x01
28 #define CMD_DIGITAL_WRITE 0x02
29 #define CMD_ANALOG_READ_REQ 0x03
30 #define CMD_ANALOG_READ_RESP 0x04
31 #define CMD_ANALOG_WRITE 0x05
32 #define CMD_BUZZER_TONE 0x20
33 #define CMD_DHT_REQ 0x22
34 #define CMD_DHT_RESP 0x23
35
36 void setup()
37 {
38   _bt_device.begin(9600);
39   _bt_device.flush();
40   pinMode(RED_LED_PIN,OUTPUT);
41   pinMode(BLUE_LED_PIN,OUTPUT);
42   pinMode(SW1_PIN,OUTPUT);
43   pinMode(SW2_PIN,OUTPUT);
44   pinMode(BUZZER_PIN,OUTPUT);
45   pinMode(RGB_RED_PIN,OUTPUT);
46   pinMode(RGB_GREEN_PIN,OUTPUT);
47   pinMode(RGB_BLUE_PIN,OUTPUT);
48   sensorDHT11.begin();
49 }
50 void loop()
51 {
52   if (_bt_device.available(>0)
53   {
54     unsigned char c;
55     _bt_device.readBytes(&c,1);
```



```
56     if ((c=='@')&&(_bt_pos==0))
57         _bt_pos++;
58     else if (_bt_pos==1){
59         _bt_pos++;
60         _bt_cmd=c;
61     }
62     else if (_bt_pos==2){
63         _bt_pos++;
64         _bt_length=c;
65     }
66     else if ((_bt_pos>=3)&&(_bt_pos<=( _bt_length+2))){
67         _bt_data[_bt_pos-3]=c;
68         _bt_pos++;
69     }
70     else if ((_bt_pos==( _bt_length+3))&&(c=='*')){
71         if (_bt_cmd==CMD_DIGITAL_READ_REQ){
72             int pin=_bt_data[0];
73             if (pin==SW1_PIN){
74                 _bt_device.write('@');
75                 _bt_device.write((byte)CMD_DIGITAL_READ_RESP);
76                 _bt_device.write((byte)2);
77                 _bt_device.write((byte)SW1_PIN);
78                 _bt_device.write((byte)(byte)(digitalRead(SW1_PIN)));
79                 _bt_device.write('*');
80             }
81             else if (pin==SW2_PIN) {
82                 _bt_device.write('@');
83                 _bt_device.write((byte)CMD_DIGITAL_READ_RESP);
84                 _bt_device.write((byte)2);
85                 _bt_device.write((byte)SW2_PIN);
86                 _bt_device.write((byte)(byte)(digitalRead(SW2_PIN)));
87                 _bt_device.write('*');
88             }
89         }
90         else if (_bt_cmd==CMD_DIGITAL_WRITE){
91             int pin=_bt_data[0];
92             boolean value = _bt_data[1]==1? HIGH: LOW;
93             if (pin==RED_LED_PIN)
94                 digitalWrite(RED_LED_PIN,value);
95             else if (pin==BLUE_LED_PIN)
96                 digitalWrite(BLUE_LED_PIN,value);
97         }
98         else if (_bt_cmd==CMD_ANALOG_READ_REQ){
99             int pin=_bt_data[0];
100            if (pin==POT_PIN){
101                _bt_device.write('@');
102                _bt_device.write((byte)CMD_ANALOG_READ_RESP);
103                _bt_device.write((byte)3);
104                _bt_device.write((byte)POT_PIN);
105                short int _value=analogRead(POT_PIN);
106                _bt_device.write((byte)((_value&0xFF00)>>8));
107                _bt_device.write((byte)( _value&0x00FF));
108                _bt_device.write('*');
109            }
110            else if (pin==LDR_PIN) {
111                _bt_device.write('@');
112                _bt_device.write((byte)CMD_ANALOG_READ_RESP);
113                _bt_device.write((byte)3);
114                _bt_device.write((byte)LDR_PIN);
115                short int _value=analogRead(LDR_PIN);
116                _bt_device.write((byte)((_value&0xFF00)>>8));
117                _bt_device.write((byte)( _value&0x00FF));
118                _bt_device.write('*');
119            }
120        }
121        else if (_bt_cmd==CMD_ANALOG_WRITE){
122            int pin = _bt_data[0];
123            byte value = _bt_data[1];
```



```
124     if (pin==RGB_RED_PIN)
125         analogWrite(RGB_RED_PIN,value);
126     else if (pin==RGB_GREEN_PIN)
127         analogWrite(RGB_GREEN_PIN,value);
128     else if (pin==RGB_BLUE_PIN)
129         analogWrite(RGB_BLUE_PIN,value);
130 }
131 else if (_bt_cmd==CMD_DHT_REQ){
132     int pin = _bt_data[0];
133     if (pin ==DHT11_PIN) {
134         byte data[4]={0,0,0,0};
135         int temp=sensorDHT11.readTemperature();
136         int humid=sensorDHT11.readHumidity();
137         _bt_device.write('@');
138         _bt_device.write((byte)CMD_DHT_RESP);
139         _bt_device.write((byte)5);
140         _bt_device.write((byte)DHT11_PIN);
141         _bt_device.write((byte)(((temp)>>(8))));
142         _bt_device.write((byte)(temp));
143         _bt_device.write((byte)(((humid)>>(8))));
144         _bt_device.write((byte)(byte)(humid));
145         _bt_device.write('*');
146     }
147 }
148 if (_bt_cmd==CMD_BUZZER_TONE) {
149     int pin = _bt_data[0];
150     int frequency = (((int)_bt_data[1])<<8)|(_bt_data[2]);
151     int duration = (((int)_bt_data[3])<<8)|(_bt_data[4]);
152     if (pin ==BUZZER_PIN)
153         tone(BUZZER_PIN,frequency,duration);
154     delay(duration);
155     noTone(BUZZER_PIN);
156 }
157 _bt_pos=0;
158 _bt_length=0;
159 }
160 else{
161     _bt_pos=0;
162     _bt_length=0;
163 }
164 }
165 }
```



Example 3: Remote Control of a Robot

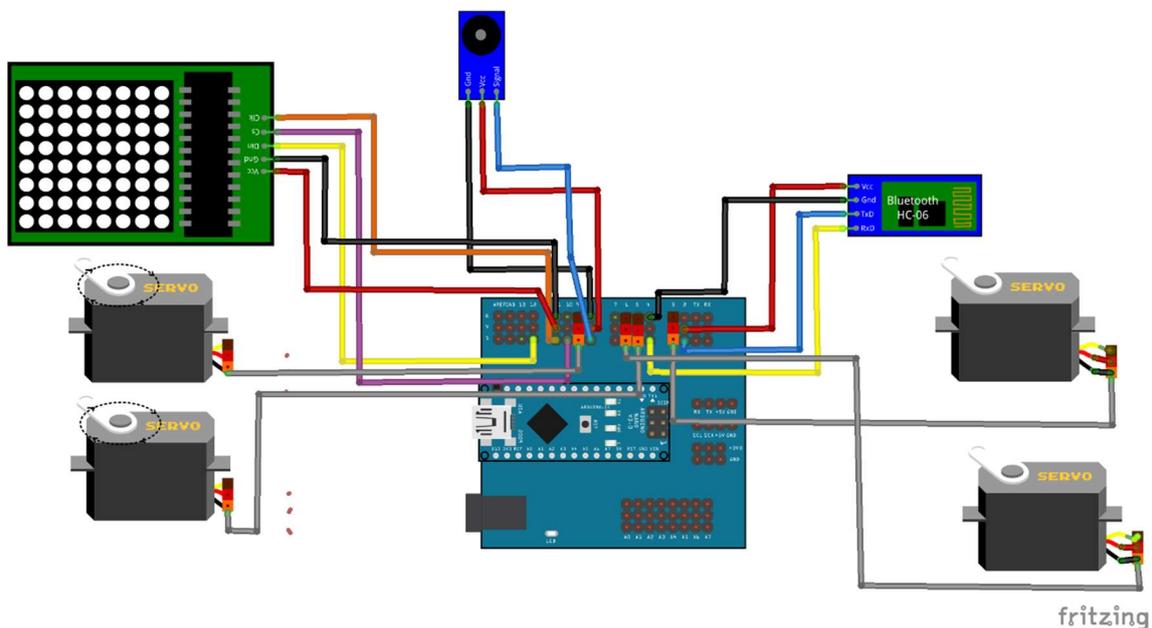
Download

https://roboticafacil.es/facilino/ai2/demos/DYOR_extension.zip

Connection diagram

The following example uses the connection diagram as shown. The circuit includes an Arduino Nano with an expansion shield, two continuous rotation servos, two standard servos, a buzzer module and a 8x8 LED Matrix module a Bluetooth module HC-05 connected as follow:

- Left and right wheel servo motors connected to pins D9 and D5.
- Left and right arm servo motors connected to pins D6 and D3.
- LED Matrix connected to pins D10 (CS), D11 (CLK), D12 (DIN).
- Buzzer module connected to pin D8.
- Bluetooth module connected to pin D2 (TX) and pin D3 (RX).



App Inventor

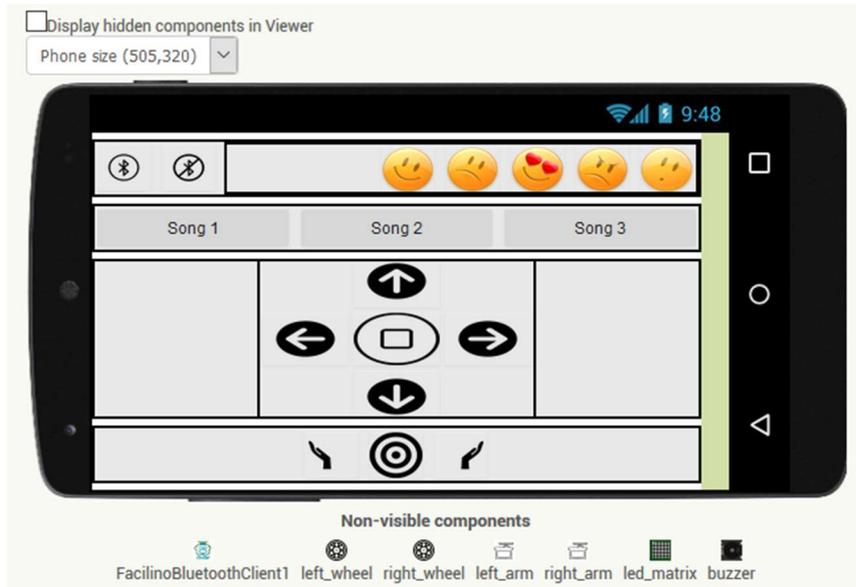
Designer instructions

- Set the 'ScreenOrientation' property of 'Screen1' component to *Landscape*.
- Create a **HorizontalArrangement** component and set the 'Width' property to fill its parent and the 'Height' property to 15%. Inside this component create a **ListPicker** component and set its 'Height' property to 13% and its 'Width' property to 8%. Rename this component to 'connect' and remove the text in the 'Text' property set the 'Image' property with an icon. You can select the icon you prefer to suit your App, i.e.: obtain it from flaticon.com, a good repository with free icons. Now, create a **Button** component and rename it as 'disconnect' and also set the 'Height', 'Width', 'Text' and 'Image' properties as you did before with the **ListPicker** component. Also create a **HorizontalArrangement** component, nested inside the previous **HorizontalArrangement** component. And inside the nested **HorizontalArrangement** component, add five **Button** components, and rename them as 'happy', 'sad', 'love', 'angry' and 'stunned'. Set the 'Image' property with the corresponding icon and remove the text in the 'Text' property and set the 'Height' and 'Width' property to 13% and 8% in the **Button** components, so that they have the aspect shown in the App UI image
- Create a **HorizontalArrangement** component and modify its properties as indicated. 'AlignHorizontal' and 'AlignVertical' properties must be set to *Center* and the 'Height' and 'Width' properties must be set to fill their parents. Then, add a **TableArrangement** component with 3 'Columns' and 3 'Rows'. Inside this **TableArrangement** component add five **Button** components as shown in the App UI image. Remove their 'Text' property, set 'Image' property with appropriate icons and rename the **Button** components as 'up',



'down', 'left', 'right' and 'stop'. Set the 'Height' and 'Width' properties of these **Button** components to 15% and 13%.

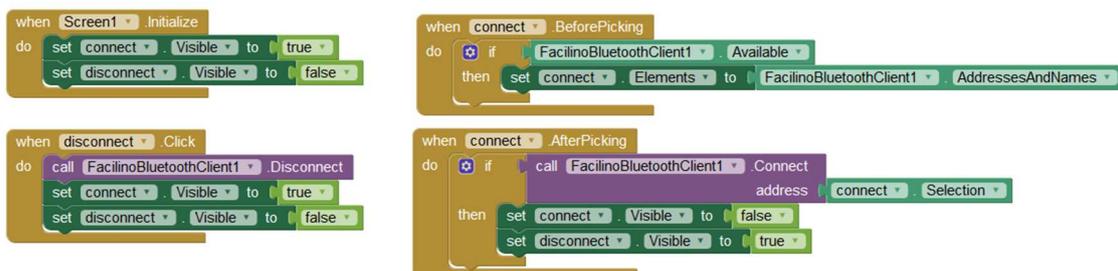
- Create a **HorizontalArrangement** component and modify its properties as indicated. 'AlignHorizontal' and 'AlignVertical' properties must be set to *Center* and the 'Width' property must be set to fill its parent. Add three **Button** components and set their 'Height' and 'Width' properties to 13% and 8%. Also remove the text in the 'Text' property and set the 'Image' property with the appropriate icon. Rename the **Button** components as 'left_arm_btn', 'right_arm_btn' and 'shoot_btn'.
- Import the **FacilinoBluetooth** AI2 extension (es.roboticafacil.com/facilino/runtime/bluetooth.aix).
- Create a **FacilinoBluetoothClient** component (default name 'FacilinoBluetoothClient1').
- Create two **ServoContBluetooth** components and rename them as 'left_wheel' and 'right_wheel' and assign the property 'FacilinoBluetoothClient' to point to the **FacilinoBluetoothClient** component. Also modify the 'Pin' property to number 9 and 5, respectively.
- Create two **ServoBluetooth** components and rename them as 'left_arm' and 'right_arm' and assign the property 'FacilinoBluetoothClient' to point to the **FacilinoBluetoothClient** component. Also modify the 'Pin' property to number 6 and 3, respectively.
- Create a **LEDMatrix8x8Bluetooth** component and rename it as 'led_matrix' and assign the property 'FacilinoBluetoothClient' to point to the **FacilinoBluetoothClient** component. Also modify the 'CLK', 'CS' and 'DIN' properties to numbers 11, 10 and 12, respectively.
- Create a **BuzzerBluetooth** component and rename it as 'buzzer' and assign the property 'FacilinoBluetoothClient' to point to the **FacilinoBluetoothClient** component. Also modify the 'Pin' property to number 8.



Blocks

Please, refer to the [REFERENCE MANUAL](#) of **FacilinoBluetooth** extension for further information on each, method, property or event.

Connection and disconnection use the 'Connect' and 'Disconnect' methods of the **FacilinoBluetoothClient** component. To list the MAC address in the **ListPicker** component, use the 'AddressesAndNames' property. To check if Bluetooth is 'Enabled' and 'Available', use the corresponding properties.





To set the movements of the robot, we call the 'Move' method of **ServoContBluetooth** components inside the 'Click' event on each **Button** component. The 'velocity' input argument of the 'Move' method is expressed in a percentage can vary between -100 and 100. So, for instance, in order to move forward, both wheel motors will rotate with a positive velocity. To stop the robot, we set both velocities to zero.

```
when up Click
do
  call left_wheel .Move
  velocity 50
  call right_wheel .Move
  velocity 50

when left Click
do
  call left_wheel .Move
  velocity -50
  call right_wheel .Move
  velocity 50

when down Click
do
  call left_wheel .Move
  velocity -50
  call right_wheel .Move
  velocity -50

when right Click
do
  call left_wheel .Move
  velocity 50
  call right_wheel .Move
  velocity -50

when stop Click
do
  call left_wheel .Move
  velocity 0
  call right_wheel .Move
  velocity 0
```

Similarly, the servos for robot arms/gripper can be moved. To do so, we call the 'Move' method of the **ServoBluetooth** components and provide the 'angle' in degrees and varies between 0 and 180.

```
when shoot_btn TouchUp
do
  call left_arm .Move
  angle 135
  call right_arm .Move
  angle 45

when shoot_btn TouchDown
do
  call left_arm .Move
  angle 45
  call right_arm .Move
  angle 135

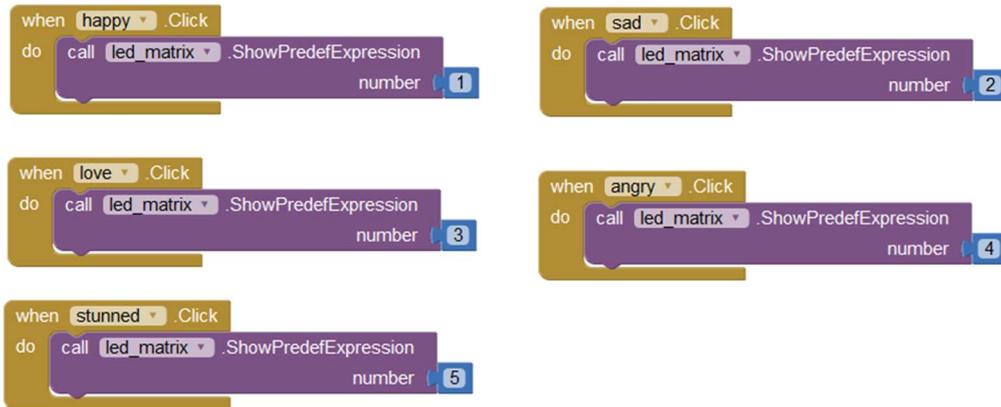
when left_arm_btn TouchUp
do
  call left_arm .Move
  angle 135

when left_arm_btn TouchDown
do
  call left_arm .Move
  angle 45

when right_arm_btn TouchUp
do
  call left_arm .Move
  angle 45

when right_arm_btn TouchDown
do
  call left_arm .Move
  angle 135
```

In order to generate expression in the LED matrix, we use the 'ShowPredefExpression' method of the **LEDMatrix8x8Bluetooth** component. We can pass a number to this method that will be interpreted, on the robot side as an expression that has been previously defined.



Similarly, in order to reproduce a song (a melody), we can call the 'Song' method of the **BuzzerBluetooth** component.



Facilino Code

The *Bluetooth definition* instruction uses pins D2 and D3, to establish the software serial connection (please, consider that RX and TX between the Bluetooth module and Arduino are crossed). Also, in the setup of the program, use the 8x8 LED matrix instruction with a blank expression to initialise the LED Matrix module.

The *Bluetooth Receive telegram* instruction decodes any received telegram. In this case, we are decoding *Servo 360*, *Servo*, *LED Matrix Predefined* and *Buzzer Melody* telegrams. We must check the pin numbers before using the corresponding instructions on each telegram.

In particular, we start with the *Servo 360* telegram, that decodes the corresponding telegram and if the pin numbers are correct, then we use the *Continuous rotation Servo* instruction to set the velocity based on the received value. Also, we can use a similar procedure to set the position of robot arms using standard servos with the *Servo* instruction.

We also add a telegram decoding case for the LED Matrix module. In this case, we need to check that all three pins match as expected and then we use a *Switch* instruction to select the appropriate expression depending on the received number. On each case, we call a function that indeed uses the *8x8 LED Matrix* instruction to set the corresponding expression.

Finally, we decode the Buzzer Melody telegram. Again, we check that the pin number is correct and also use a *Switch* instruction to reproduce the correct melody.



Setup

Bluetooth definition 

Baud rate 9600

RX  Digital PIN D2

TX  Digital PIN D3

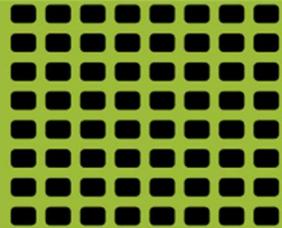
8x8 LED matrix 

CS  Digital PIN D10

DIN  Digital PIN D12

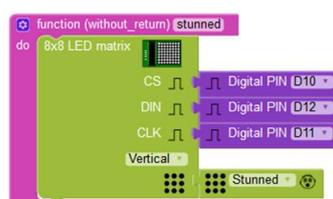
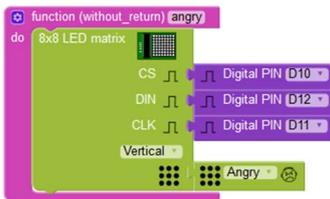
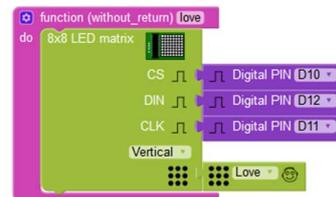
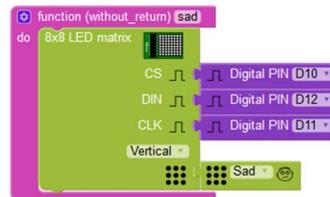
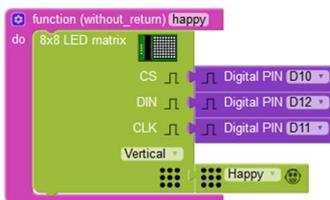
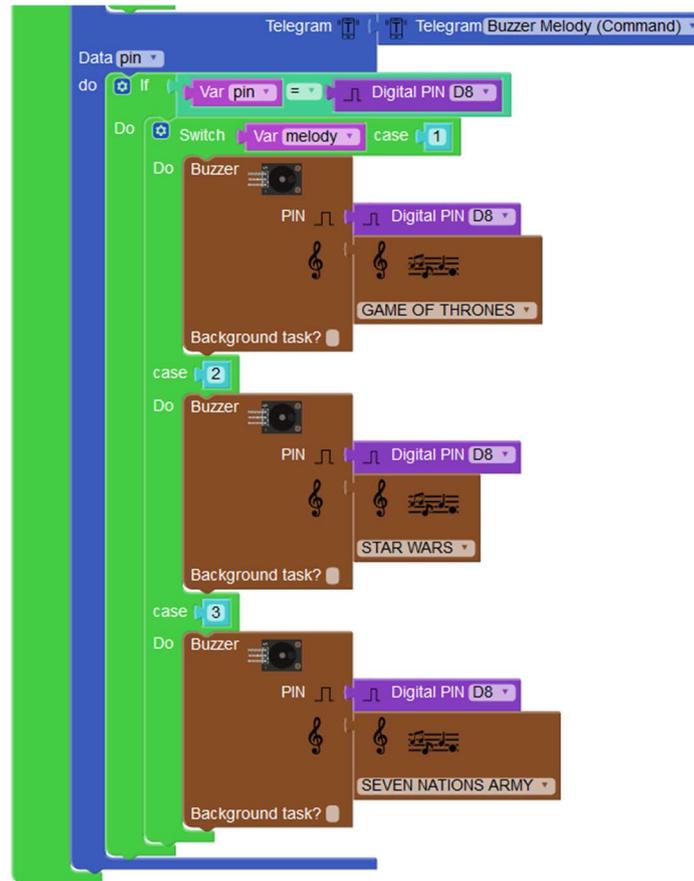
CLK  Digital PIN D11

Vertical 





```
Loop
  Bluetooth Receive telegram
  Telegram Telegram Servo 360? (Command)
  Data pin
  do
    If Var pin = Digital PIN D9
    Do Continuous rotation Servo
      PIN Digital PIN D9
      Speed % (-100~100) Var value
    else if Var pin = Digital PIN D5
    Do Continuous rotation Servo
      PIN Digital PIN D5
      Speed % (-100~100) Var value
  Telegram Telegram Servo (Command)
  Data pin
  do
    If Var pin = Digital PIN D6
    Do Servo
      PIN Digital PIN D6
      Degrees (0~180) Var value
    else if Var pin = Digital PIN D3
    Do Servo
      PIN Digital PIN D3
      Degrees (0~180) Var value
  Telegram Telegram LED Matrix Predefined (Command)
  Data pin
  do
    If Var pin_clk = Digital PIN D11
    Do If Var pin_cs = Digital PIN D10
    Do If Var pin_din = Digital PIN D12
    Do Switch Var number case 1
    Do happy
    case 2
    Do sad
    case 3
    Do love
    case 4
    Do angry
    case 5
    Do stunned
```



Arduino Code

The following code decodes two types of telegrams, *Servo*, *Servo Continuous*, *Buzzer Melody* and *LED Matrix Expression*, with commands 0x10 and 0x11, 0x21, 0x51, respectively. See [PROTOCOL MANUAL](#) for further details. First, we initialise the Bluetooth module with 9600 bauds, based on the **SoftwareSerial** library and set the pin modes to the corresponding mode as well as the servo attach. Then, in the loop function, we implement the actual telegram decoding, so in line 108 checks the type of telegram received, obtaining meaningful information from the telegram data such as pin and value. Depending on the receive telegram, the code between lines 108 and 172 performs the corresponding action.

This Arduino code also includes three melodies to reproduce Game of Thrones (HBO Serie), Imperial March (Star Wars) and Seven Nations Army (White Stripes) themes. It also includes the code to show expressions in a LED 8x8 Matrix.



```
1  #include <SoftwareSerial.h>
2  #include <Servo.h>
3
4  #define BT_TX_PIN 2 //TX pin of bluetooth module
5  #define BT_RX_PIN 3 //RX pin of bluetooth module
6  #define LEFT_WHEEL_PIN 9
7  #define RIGHT_WHEEL_PIN 5
8  #define LEFT_ARM_PIN 6
9  #define RIGHT_ARM_PIN 3
10 #define CLK_PIN 11
11 #define CS_PIN 10
12 #define DIN_PIN 12
13 #define BUZZER_PIN 8
14
15 SoftwareSerial _bt_device(BT_TX_PIN,BT_RX_PIN);
16 Servo leftWheel;
17 Servo rightWheel;
18 Servo leftArm;
19 Servo rightArm;
20
21 int _bt_pos=0;
22 unsigned char _bt_cmd=0;
23 int _bt_length=0;
24 unsigned char _bt_data[255];
25
26 #define CMD_SERVO 0x10
27 #define CMD_SERVO_CONT 0x11
28 #define CMD_BUZZER_MELODY 0x21
29 #define CMD_LED_MATRIX_PREDEF_EXPR 0x51
30
31 const uint16_t GAME_THRONES[] =
32 {1568,562,1047,562,1245,93,1397,93,1568,375,1047,375,1245,93,1397,93,1175,1875
33 ,1397,562,932,562,1245,93,1175,93,1397,375,932,562,1245,93,1175,93,1047,750};
34 const uint16_t IMPERIAL_MARCH[] =
35 {440,375,440,375,440,375,349,281,523,93,440,375,349,281,523,93,440,750,659,375
36 ,659,375,659,375,698,281,523,93,415,375,349,281,523,93,440,750,880,375,440,281
37 ,440,93,880,375,831,281,784,93,740,93,698,93,740,187,0,187,446,281,622,375,587
38 ,281,554,93,523,93,494,93,523,187,0,187,349,187,415,375,349,281,392,93,523,375
39 ,440,281,523,93,659,750,880,375,440,281,440,93,880,375,831,281,784,93,740,93,6
40 98,93,740,187,0,187,446,281,622,375,587,281,554,93,523,93,494,93,523,187,0,187
41 ,330,187,415,375,330,281,523,93,440,375,349,281,523,93,440,750};
42 const uint16_t SEVEN_NATIONS_ARMY[] =
43 {330,759,330,252,392,378,330,125,0,252,294,252,262,1012,245,1012,330,759,330,2
44 52,392,378,330,125,0,252,294,252,262,506,294,252,262,252,245,1012,330,759,330,
45 252,392,378,330,125,0,252,294,252,262,1012,245,1012};
46
47 void LEDMatrix_init(int cs, int din, int clk);
48 void writeRow(int cs, int din, int clk, int row, int data);
49 void maxAll (int cs, int din, int clk, int reg, int col);
50 void putByte (int din, int clk, int data);
51 void expression(int cs, int din, int clk, int col1, int col2, int col3, int
52 col4, int col5, int col6, int col7, int col8);
53 void playMelody(int pin,const uint16_t* melody, int length);
54 void happy ();
55 void sad ();
56 void love ();
57 void angry ();
58 void stunned ();
59
60 void setup()
61 {
62   _bt_device.begin(9600);
63   _bt_device.flush();
64   pinMode(BUZZER_PIN,OUTPUT);
65   pinMode(CLK_PIN,OUTPUT);
66   pinMode(CS_PIN,OUTPUT);
67   pinMode(DIN_PIN,OUTPUT);
68   //Set the LED Matrix component
```



```
69 maxAll(CS_PIN,DIN_PIN,CLK_PIN,11,7);
70 maxAll(CS_PIN,DIN_PIN,CLK_PIN,9,0);
71 maxAll(CS_PIN,DIN_PIN,CLK_PIN,12,1);
72 maxAll(CS_PIN,DIN_PIN,CLK_PIN,15,0);
73 for (int i = 1; i <= 8; i++)
74     maxAll(CS_PIN,DIN_PIN,CLK_PIN,i,0);
75 maxAll(CS_PIN,DIN_PIN,CLK_PIN,10,15);
76 expression(CS_PIN,DIN_PIN,CLK_PIN,0,0,0,0,0,0,0,0);
77
78 leftWheel.attach(LEFT_WHEEL_PIN);
79 delay(300);
80 rightWheel.attach(RIGHT_WHEEL_PIN);
81 delay(300);
82 leftArm.attach(LEFT_ARM_PIN);
83 delay(300);
84 rightArm.attach(RIGHT_ARM_PIN);
85 delay(300);
86 }
87 void loop()
88 {
89     if (_bt_device.available()>0)
90     {
91         unsigned char c;
92         _bt_device.readBytes(&c,1);
93         if ((c=='@')&&(_bt_pos==0))
94             _bt_pos++;
95         else if (_bt_pos==1){
96             _bt_pos++;
97             _bt_cmd=c;
98         }
99         else if (_bt_pos==2){
100            _bt_pos++;
101            _bt_length=c;
102        }
103        else if ((_bt_pos>=3)&&(_bt_pos<=( _bt_length+2))){
104            _bt_data[_bt_pos-3]=c;
105            _bt_pos++;
106        }
107        else if ((_bt_pos==( _bt_length+3))&&(c=='*')){
108            if (_bt_cmd==CMD_SERVO_CONT){
109                int pin = _bt_data[0];
110                byte value = _bt_data[1];
111                if (pin==LEFT_WHEEL_PIN)
112                    leftWheel.write (((value*90)/100+90));
113                else if (pin==RIGHT_WHEEL_PIN)
114                    rightWheel.write (((value*90)/100+90));
115            }
116            if (_bt_cmd==CMD_SERVO){
117                int pin = _bt_data[0];
118                byte value = _bt_data[1];
119                if (pin==LEFT_ARM_PIN)
120                    leftArm.write(value);
121                else if (pin==RIGHT_ARM_PIN)
122                    rightArm.write(value);
123            }
124            if (_bt_cmd==CMD_LED_MATRIX_PREDEF_EXPR){
125                int pin_clk = _bt_data[0];
126                int pin_cs = _bt_data[1];
127                int pin_din = _bt_data[2];
128                byte number = _bt_data[3];
129                if ((pin_clk==CLK_PIN)&&(pin_cs==CS_PIN)&&(pin_din==DIN_PIN)){
130                    switch (number)
131                    {
132                        case 1:
133                            happy();
134                            break;
135                        case 2:
136                            sad();
```



```
137         break;
138     case 3:
139         love();
140         break;
141     case 4:
142         angry();
143         break;
144     case 5:
145         stunned();
146         break;
147     }
148 }
149 }
150 if (_bt_cmd==CMD_BUZZER_MELODY){
151     int pin = _bt_data[0];
152     byte melody = _bt_data[1];
153     if (pin==BUZZER_PIN){
154         switch (melody)
155         {
156             case 1:
157
158 playMelody(BUZZER_PIN, GAME_THRONES, sizeof(GAME_THRONES)/(2*sizeof(uint16_t)));
159                 break;
160             case 2:
161
162 playMelody(BUZZER_PIN, IMPERIAL_MARCH, sizeof(IMPERIAL_MARCH)/(2*sizeof(uint16_t)
163 ));
164                 break;
165             case 3:
166
167 playMelody(BUZZER_PIN, SEVEN_NATIONS_ARMY, sizeof(SEVEN_NATIONS_ARMY)/(2*sizeof(
168 uint16_t)));
169                 break;
170         }
171     }
172 }
173     _bt_pos=0;
174     _bt_length=0;
175 }
176 else{
177     _bt_pos=0;
178     _bt_length=0;
179 }
180 }
181 }
182
183 void LEDMatrix_init(int cs, int din, int clk) {
184     maxAll(cs,din,clk,11,7);
185     maxAll(cs,din,clk,9,0);
186     maxAll(cs,din,clk,12,1);
187     maxAll(cs,din,clk,15,0);
188     for (int i = 1; i <= 8; i++)
189         maxAll(cs,din,clk,i, 0);
190     maxAll(cs,din,clk,10,15);
191 }
192
193 void writeRow(int cs, int din, int clk, int row, int data) {
194     digitalWrite(cs,LOW);
195     putByte(din,clk,row);
196     putByte(din,clk,data);
197     digitalWrite(cs,LOW);
198     digitalWrite(cs,HIGH);
199 }
200
201 void maxAll (int cs, int din, int clk, int reg, int col) {
202     digitalWrite(cs,LOW);
203     putByte(din,clk,reg);
204     putByte(din,clk,col);
```



```
205     digitalWrite(cs,LOW);
206     digitalWrite(cs,HIGH);
207 }
208 void putByte (int din, int clk, int data) {
209     byte i = 8;
210     byte mask;
211     while(i > 0) {mask = 0x01 << (i - 1);
212         digitalWrite(clk,LOW);
213         if (data & mask)
214         {
215             digitalWrite(din,HIGH);
216         }else{
217             digitalWrite(din,LOW);
218         }
219         digitalWrite(clk,HIGH);
220         --i;
221     }
222 }
223
224 void expression(int cs, int din, int clk, int col1, int col2, int col3, int
225 col4, int col5, int col6, int col7, int col8) {
226     writeRow(cs,din,clk,1, col1);
227     writeRow(cs,din,clk,2, col2);
228     writeRow(cs,din,clk,3, col3);
229     writeRow(cs,din,clk,4, col4);
230     writeRow(cs,din,clk,5, col5);
231     writeRow(cs,din,clk,6, col6);
232     writeRow(cs,din,clk,7, col7);
233     writeRow(cs,din,clk,8, col8);
234 }
235
236 void playMelody(int pin,const uint16_t* melody, int length)
237 {
238     unsigned int note;
239     unsigned long duration;
240     uint16_t* melody_ptr=(uint16_t*)melody;
241     for (int i=0;i<length;i++)
242     {
243         note=*melody_ptr++;
244         duration=*melody_ptr++;
245         tone(pin,note,duration);
246         delay(duration);
247         noTone(pin);
248     }
249 }
250
251 void happy () {
252     expression(10,12,11,12,24,48,48,48,48,24,12);
253 }
254
255 void sad () {
256     expression(10,12,11,48,24,12,12,12,12,24,48);
257 }
258
259 void love () {
260     expression(10,12,11,12,30,62,124,124,62,30,12);
261 }
262
263 void angry () {
264     expression(10,12,11,2,68,36,16,16,36,68,2);
265 }
266
267 void stunned () {
268     expression(10,12,11,0,0,0,24,24,0,0,0);
269 }
```



Example 4: Remote Configuration of a Device

Remark considerations

This example represents part of the code used for Reespirator device, an artificial breathing machine develop during the COVID-19 crisis. Here, we just show how to remotely set the configuration of the device. So, the underlying idea is that we use this App to remotely set a set of given parameters. All the code regarding with the electronic control is not included here for simplification.

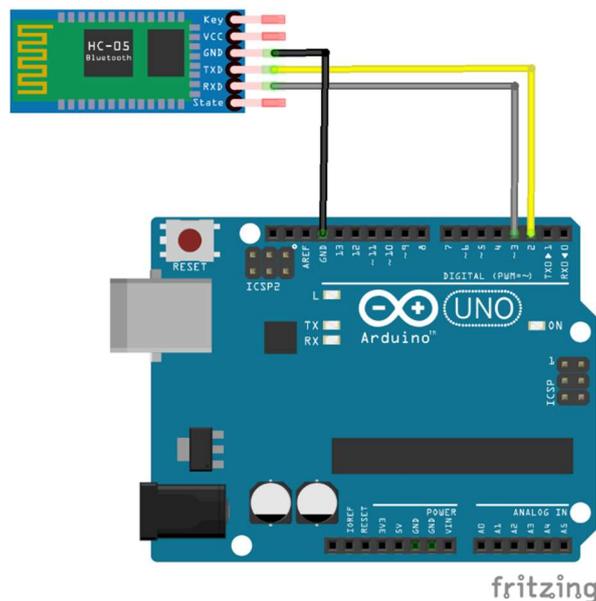
Download

https://roboticafacil.es/facilino/ai2/demos/Reespirator_demo.zip

Connection diagram

The following example uses the connection diagram as shown. The circuit includes an Arduino Uno with a Bluetooth module HC-05 connected as follow:

- Bluetooth module connected to pin D2 (TX) and pin D3 (RX).



App Inventor

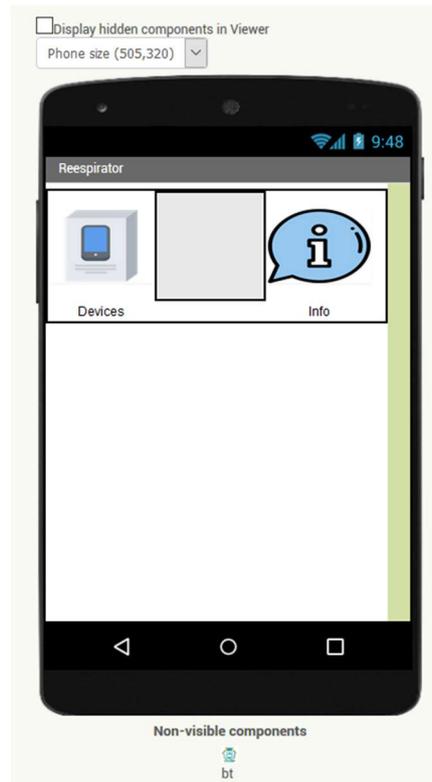
For this App, we have several screens. One of the common problems when dealing with multiple screens with the standard **BluetoothClient** component included in App Inventor 2 is that it gets disconnected. This is easily solved with the **FacilinoBluetoothClient** component, due to its 'Reconnect' method. This feature will be shown in this example, through different screens.

Designer instructions are not given for simplicity, just the list of main components names, types and a short description on each screen.

Screen1 Designer

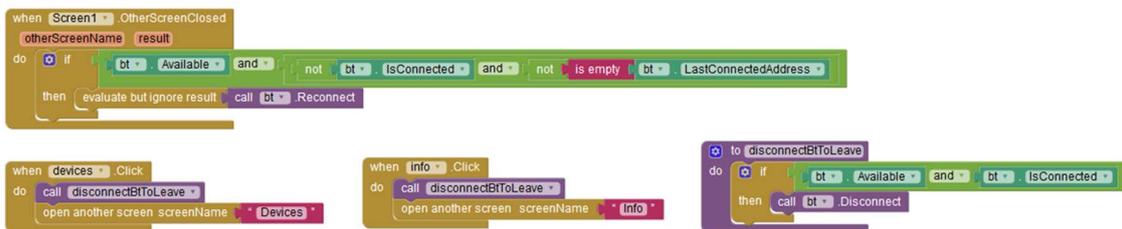
A simple UI with two **Button** components and a **FacilinoBluetoothClient** component. **Button** components are used to switch to other screens.

Screen1		
Name	Component Type	Description
devices	Button	Click on it to switch to 'Device' screen.
info	Button	Click on it to switch to 'Info' screen.
bt	FacilinoBluetoothClient	Bluetooth Client.



Screen1 Blocks

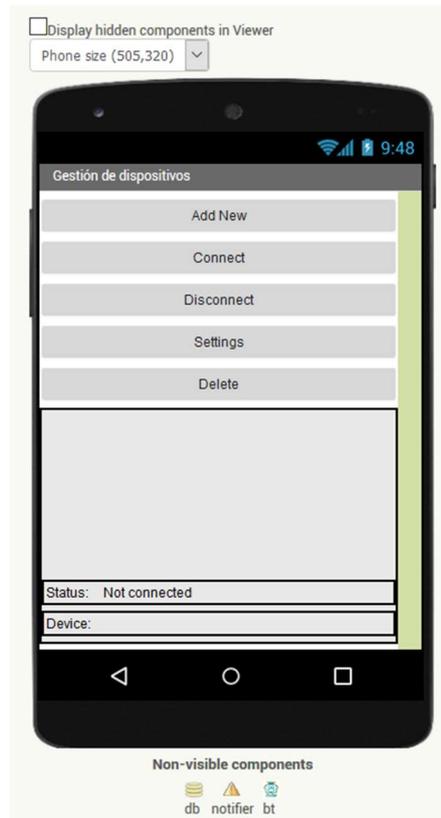
When each of the **Button** component is click ('Click' event), we call the procedure to disconnect the bluetooth device and then we open the other screen. The **FacilinoBluetoothClient** is included here to keep connection, although it is not strictly necessary. In addition to this, when that screen is closed, we try to reconnect again.



Devices Designer

A UI with several **Button** and **ListPicker** components to manage all connection issues. It also includes a **FacilinoBluetoothClient**, a **TinyDB** and **Notifier** components.

Devices		
Name	Component Type	Description
addNew	ListPicker	To show available devices and register the one to connect to.
connect	Button	To connect to a device within the list of previously registered devices.
disconnect	Button	To disconnect to the currently connected device.
settings	Button	Click on it to switch to 'Settings' screen.
delete	ListPicker	To show the list of registered devices and remove it from the list.
status	Label	To show the connection status.
connectedDevice	Label	To display the MAC and name of the currently connected device.
db	TinyDB	To register devices.
notifier	Notifier	To show messages to the user.
bt	FacilinoBluetoothClient	Bluetooth Client.



Devices Blocks

First, we show all event methods related with the actions to take when thrown.

```
initialize glob(devices to create empty list

when Devices Initialize
do if bt Available and not is empty bt LastConnectedAddress
then if call bt Reconnect
then call setConnectMode
device bt LastConnectedAddress
set global devices to call db GetValue
tag devices
valueIfTagNotThere create empty list

when addNew BeforePicking
do if bt Available
then set addNew Elements to bt AddressesAndNames

when addNew AfterPicking
do call connectNewBt
newDevice addNew Selection
call disconnectBtToLeave
open another screen screenName Settings

when connect AfterPicking
do call connectBt
device connect Selection

when notifier AfterChoosing
choice
do if get choice == Yes
then remove list item list get global devices
index index in list thing delete Selection
list get global devices
call notifier ShowAlert
notice Device removed
if delete Selection == bt LastConnectedAddress
then call disconnectBt

when Devices BackPressed
do call disconnectBtToLeave
close screen

when connect BeforePicking
do set connect Elements to get global devices

when settings Click
do call disconnectBtToLeave
open another screen screenName Settings

when disconnect Click
do call disconnectBt

when Devices OtherScreenClosed
otherScreenName result
do evaluate but ignore result call bt Reconnect

when delete AfterPicking
do if is in list? thing delete Selection
list get global devices
then call notifier ShowChooseDialog
message Are you sure?
title delete Selection
button1Text Yes
button2Text No
cancelable false

when delete BeforePicking
do set delete Elements to get global devices
```



Then, we show all related procedures containing a set of instructions to execute as a consequence event action:

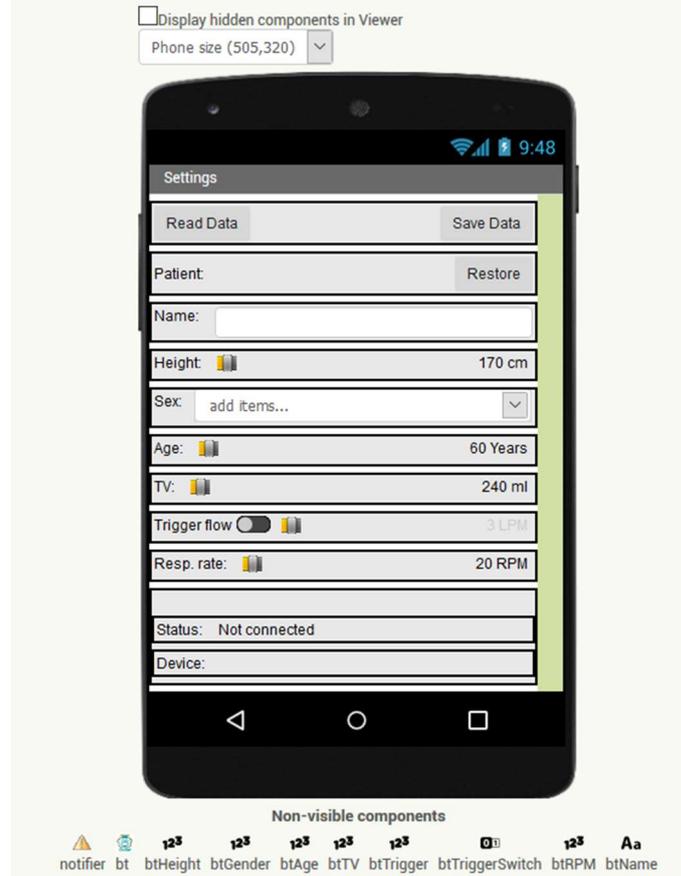
Settings Designer

This UI contains a combination of **Button**, **TextBox**, **Slider**, **Label**, **Spinner**, **Switch** components to display patient data as well as to read, save and restore data. It also includes a **FacilinoBluetoothClient** with a set of **IntVariableBluetooth**, **BooleanVariableBluetooth** and **StringVariableBluetooth** components to read or write data to the remote device.

Settings		
Name	Component Type	Description
read	Button	To read setting data from the remote device.
save	Button	To save current settings on the remote device.
restore	Button	To restore default values.
name	TextBox	Name of the patient.
height	Slider	Height of the patient (to modify with a slider).
heightLbl	Label	Height of the patient in cm.
gender	Spinner	Gender of the patient: male or female.
age	Slider	Age of the patient (to modify with a slider).
ageLbl	Label	Age of the patient in years.
TV	Slider	Tidal volume (to modify with a slider).
TVLbl	Label	Tidal volume in ml
triggerSwitch	Switch	Switch to enable/disable the trigger flow option.
trigger	Slider	Trigger flow (to modify with a slider).
triggerLbl	Label	Trigger flow in LPM.
rpm	Slider	Respiratory rate (to modify with a slider).
rpmLbl	Label	Respiratory rate in RMP
status	Label	To show the connection status.
connectedDevice	Label	To display the MAC and name of the currently connected device.
notifier	Notifier	To show messages to the user.
bt	FacilinoBluetoothClient	Bluetooth Client.
btHeight	IntVariableBluetooth	To read and write patient's height through bluetooth.
btGender	IntVariableBluetooth	To read and write patient's gender through bluetooth.
btAge	IntVariableBluetooth	To read and write patient's age through bluetooth.
btTV	IntVariableBluetooth	To read and write patient's tidal volume through bluetooth.
btTrigger	IntVariableBluetooth	To read and write patient's trigger flow through bluetooth.



btTriggerSwitch	BooleanVariableBluetooth	To read and write patient's trigger flow status through bluetooth.
btRPM	IntVariableBluetooth	To read and write patient's respiratory rate through bluetooth.
btName	StringVariableBluetooth	To read and write patient's gender through bluetooth.



Settings Blocks

First, let's declare some global variables containing the parameters to set.



Then, let's handle connection issues. When the screen is initialised, we reconnect the Bluetooth device, disconnect when the back button is pressed. It also checks whether or not the data has been modified, to save it before leaving. Most of the visual components of the screen are disabled if the device is not connected.



```
when Settings.Initialize
do
  if [bt . Available] and [not [bt . IsConnected] and [not [is empty [bt . LastConnectedAddress]]]
  then
    evaluate but ignore result call [bt .Reconnect]
    call [setConnectMode]
      device [bt . LastConnectedAddress]
    call [readData]
  else
    call [setDisconnectMode]

when Settings.BackPressed
do
  if [get global saved] or [get global never_saved]
  then
    call [disconnectBtToLeave]
    close screen
  else
    call [notifier .ShowChooseDialog]
      message [Settings not saved. Are you sure?]
      title [Warning]
      button1Text [Yes]
      button2Text [No]
      cancelable [false]
```

```
when notifier.AfterChoosing
choose
do
  if [get choice] = [Yes]
  then
    call [disconnectBtToLeave]
    close screen
```

```
to [setConnectMode] device
do
  set [status . Text] to [Connected]
  set [connectedDevice . Text] to [get device]
  set [read . Enabled] to [true]
  set [save . Enabled] to [true]
  set [restore . Enabled] to [true]
  set [name . Enabled] to [true]
  set [height . ThumbEnabled] to [true]
  set [heightLbl . TextColor] to [black]
  set [age . ThumbEnabled] to [true]
  set [ageLbl . TextColor] to [black]
  set [TV . ThumbEnabled] to [true]
  set [TVLbl . TextColor] to [black]
  if [triggerSw itch . On]
  then
    set [trigger . ThumbEnabled] to [true]
    set [triggerLbl . TextColor] to [black]
  else
    set [trigger . ThumbEnabled] to [false]
    set [triggerLbl . TextColor] to [gray]
  set [rpm . ThumbEnabled] to [true]
  set [rpmLbl . TextColor] to [black]
```

```
to [setDisconnectMode]
do
  set [status . Text] to [Disconnected]
  set [connectedDevice . Text] to [ ]
  set [read . Enabled] to [false]
  set [save . Enabled] to [false]
  set [restore . Enabled] to [false]
  set [name . Enabled] to [false]
  set [height . ThumbEnabled] to [false]
  set [heightLbl . TextColor] to [gray]
  set [age . ThumbEnabled] to [false]
  set [ageLbl . TextColor] to [gray]
  set [TV . ThumbEnabled] to [false]
  set [TVLbl . TextColor] to [gray]
  set [trigger . ThumbEnabled] to [false]
  set [triggerLbl . TextColor] to [gray]
  set [rpm . ThumbEnabled] to [false]
  set [rpmLbl . TextColor] to [gray]
```

These three 'Click' events call procedures that actually retrieve, save or restore default values of data:

```
when [read] .Click
do
  call [readData]
  set [global saved] to [true]
```

```
when [save] .Click
do
  call [saveData]
  set [global saved] to [true]
  set [global never_saved] to [false]
```

```
when [restore] .Click
do
  call [restoreData]
  set [global saved] to [false]
```

These are the procedures used in the previous 'Click' events.



```
to readData
do
  if [bt . Available] and [bt . IsConnected]
  then
    call [btName] . Update
    call [btHeight] . Update
    call [btGender] . Update
    call [btAge] . Update
    call [btTV] . Update
    call [btTriggerSwitch] . Update
    call [btTrigger] . Update
    call [btRPM] . Update
```

```
to saveData
do
  if [bt . Available] and [bt . IsConnected]
  then
    call [btName] . Set
      value [name] . Text
    call [btHeight] . Set
      value [get global height]
    call [btGender] . Set
      value [get global gender]
    call [btAge] . Set
      value [get global age]
    call [btTV] . Set
      value [get global TV]
    call [btTriggerSwitch] . Set
      value [get global triggerSwitch]
    call [btTrigger] . Set
      value [get global trigger]
    call [btRPM] . Set
      value [get global rpm]
```

```
to restoreData
do
  set [global triggerSwitch] to false
  set [global trigger] to 3
  set [global height] to 170
  set [global age] to 50
  set [global gender] to 0
  set [global rpm] to 20
  set [global TV] to 240
  set [global weight] to 70
  set [global weight0] to 50
  set [name] . Text to ""
  set [height] . ThumbPosition to [get global height]
  set [heightLbl] . Text to [join [get global height]
    " cm"]
  set [gender] . SelectionIndex to [get global gender]
  set [age] . ThumbPosition to [get global age]
  set [ageLbl] . Text to [join [get global age]
    " Years"]
  set [TV] . ThumbPosition to [get global TV]
  set [TVLbl] . Text to [join [get global TV]
    " ml"]
  set [triggerSwitch] . On to [get global triggerSwitch]
  if [triggerSwitch] . On
  then
    set [trigger] . ThumbEnabled to true
    set [triggerLbl] . TextColor to black
    set [trigger] . ThumbPosition to [get global trigger]
    set [triggerLbl] . Text to [join [get global trigger]
      " LRM"]
  else
    set [trigger] . ThumbEnabled to false
    set [triggerLbl] . TextColor to gray
  set [rpm] . ThumbPosition to [get global rpm]
```

When 'read' Button component 'Click' event is thrown, it calls the 'readData' procedure below and the 'Received' event of each bluetooth component is expected to be thrown upon request.

```
when [btHeight] . Received
value
do
  set [global height] to [round] [get value]
  set [height] . ThumbPosition to [get global height]
  set [heightLbl] . Text to [join [get global height]
    " cm"]
```

```
when [btName] . Received
value
do
  set [name] . Text to [get value]
```

```
when [btAge] . Received
value
do
  set [global age] to [round] [get value]
  set [age] . ThumbPosition to [get global age]
  set [ageLbl] . Text to [join [get global age]
    " Years"]
```

```
when [btRPM] . Received
value
do
  set [global rpm] to [round] [get value]
  set [rpm] . ThumbPosition to [get global rpm]
  set [rpmLbl] . Text to [join [get global rpm]
    " RPM"]
```

```
when [btTriggerSwitch] . Received
value
do
  set [global triggerSwitch] to [get value]
  set [triggerSwitch] . On to [get global triggerSwitch]
```

```
when [btTrigger] . Received
value
do
  set [global trigger] to [round] [get value]
  if [triggerSwitch] . On
  then
    set [trigger] . ThumbEnabled to true
    set [triggerLbl] . TextColor to black
  else
    set [trigger] . ThumbEnabled to false
    set [triggerLbl] . TextColor to gray
  set [trigger] . ThumbPosition to [get global trigger]
  set [triggerLbl] . Text to [join [get global trigger]
    " LRM"]
```

```
when [btTV] . Received
value
do
  set [global TV] to [round] [get value]
  set [TV] . ThumbPosition to [get global TV]
  set [TVLbl] . Text to [join [get global TV]
    " ml"]
```

```
when [btGender] . Received
value
do
  set [gender] . SelectionIndex to [get value]
  set [global gender] to [get value]
```



The following events (and procedure) modify the values of global variables based on the user interaction:

```

when rpm .PositionChanged
thumbPosition
do
set global rpm to round get thumbPosition
set rpmLbl .Text to join get global rpm
" RPM "
set global saved to false

when age .PositionChanged
thumbPosition
do
set global age to get thumbPosition
set ageLbl .Text to join round get global age
" Años "
set global saved to false

when TV .PositionChanged
thumbPosition
do
set global TV to round get thumbPosition
set TVLbl .Text to join get global TV
" ml "
set global saved to false

when gender .AfterSelecting
selection
do
set global gender to gender .SelectionIndex
call VTidalEstimate
set global saved to false

when trigger .PositionChanged
thumbPosition
do
set global trigger to round get thumbPosition
set triggerLbl .Text to join get global trigger
" LPM "
set global saved to false

when triggerSwitch .Changed
do
set global triggerSwitch to triggerSwitch .On
if triggerSwitch .On
then
set trigger .ThumbEnabled to true
set triggerLbl .TextColor to black
else
set trigger .ThumbEnabled to false
set triggerLbl .TextColor to gray
set global saved to false

to VTidalEstimate
do
if get global gender = 0
then
set global weight0 to 50
else
set global weight0 to 45.5
set global weight to 0.91 * get global height - 152.4 + get global weight0
set global TV to round get global weight * 7
set TV .ThumbPosition to get global TV
set TVLbl .Text to join get global TV
" ml "

```

Info Designer

A very simple UI to show info about this project. The **FacilinoBluetoothClient** is included here to keep connection, although it is not strictly necessary.

Info		
Name	Component Type	Description
bt	FacilinoBluetoothClient	Bluetooth Client.



Info Blocks

This code handles reconnection and disconnection.

```
when info.Initialize
do
  set CampoDeTexto1.Text to join ["Open-source project to design low-cost automatic...", "Project created by people for people."]
  if bt.Available and not bt.IsConnected and not isEmpty(bt.LastConnectedAddress)
  then evaluate but ignore result call bt.Reconnect

when info.BackPressed
do
  call disconnectBtToLeave
  close screen

to disconnectBtToLeave
do
  if bt.Available and bt.IsConnected
  then call bt.Disconnect
```

Arduino Code

The following code decodes two types of telegrams, *Boolean Variable*, *Integer Variable* and *String Variable*, with commands 0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x89, 0x8A and 0x8B. See [PROTOCOL MANUAL](#) for further details. First, we initialise the Bluetooth module with 9600 bauds, based on the **SoftwareSerial** library and initialise some default values for variables. Then, in the loop function, we implement the actual telegram decoding, so in line 97 checks the type of telegram received, obtaining meaningful information from the telegram data such as pin and value. Depending on the receive telegram, the code between lines 97 and 293 performs the corresponding action.

This Arduino code include all needed code to store variables when a new setting is received as well as the code to provide requested data and sending telegrams upon request. These telegrams will be received by the App and the corresponding variable and UI component updated.



```
1 #include <SoftwareSerial.h>
2
3 #define BT_TX_PIN 2 //TX pin of bluetooth module
4 #define BT_RX_PIN 3 //RX pin of bluetooth module
5
6 //Maximum number of variables to store for receive/transmitting configuration
7 #define MAX_BOOLs 1
8 #define MAX_INTs 20
9 #define MAX_FLOATs 0
10 #define MAX_STRINGS 1
11
12 #define PROTOCOL_DEBUG //Uncomment for debug
13
14 #define CMD_BOOLEAN_VAR_WRITE_REQ 0x80
15 #define CMD_BOOLEAN_VAR_READ_REQ 0x81
16 #define CMD_BOOLEAN_VAR_READ_RESP 0x82
17 #define CMD_INT_VAR_WRITE_REQ 0x83
18 #define CMD_INT_VAR_READ_REQ 0x84
19 #define CMD_INT_VAR_READ_RESP 0x85
20 #define CMD_FLOAT_VAR_WRITE_REQ 0x86
21 #define CMD_FLOAT_VAR_READ_REQ 0x87
22 #define CMD_FLOAT_VAR_READ_RESP 0x88
23 #define CMD_STRING_VAR_WRITE_REQ 0x89
24 #define CMD_STRING_VAR_READ_REQ 0x8A
25 #define CMD_STRING_VAR_READ_RESP 0x8B
26
27 bool boolVars[MAX_BOOLs];
28 int intVars[MAX_INTs];
29 float floatVars[MAX_FLOATs];
30 String stringVars[MAX_STRINGS];
31
32 //Position of boolean variables
33 #define VAR_TRIGGER_SWITCH 0
34 //Position of integer variables
35 #define VAR_HEIGHT 0
36 #define VAR_GENDER 1
37 #define VAR_AGE 2
38 #define VAR_VTIDAL 3
39 #define VAR_TRIGGER 4
40 #define VAR_RPM 5
41 //Position of string variables
42 #define VAR_NAME 0
43
44 //Variables related with the bluetooth protocol
45 SoftwareSerial _bt_device(BT_TX_PIN,BT_RX_PIN);
46 int _bt_pos=0;
47 unsigned char _bt_cmd=0;
48 int _bt_length=0;
49 unsigned char _bt_data[255];
50
51 void parseBtTelegram();
52
53 void setup()
54 {
55     //Initialise bluetooth device
56     Serial.begin(115200);
57     _bt_device.begin(9600);
58     _bt_device.flush();
59     intVars[VAR_HEIGHT]=165;
60     intVars[VAR_GENDER]=1;
61     intVars[VAR_AGE]=15;
62     intVars[VAR_VTIDAL]=240;
63     intVars[VAR_TRIGGER]=5;
64     intVars[VAR_RPM]=19;
65     stringVars[VAR_NAME]="Pepe";
66 }
67
68 void loop()
```



```
69 {
70 //IMPORTANT: To properly work, we must ensure that the rest of the code
71 // allows this routine to execute regularly, otherwise, it won't be
72 // responsive
73 while (_bt_device.available()>0)
74     parseBtTelegram();
75 }
76
77 void parseBtTelegram()
78 {
79     unsigned char c;
80     _bt_device.readBytes(&c,1);
81 #ifndef PROTOCOL_DEBUG
82     Serial.print((byte)c);
83     Serial.print(" ");
84 #endif
85     if ((c=='@')&&(_bt_pos==0))
86         _bt_pos++;
87     else if (_bt_pos==1) {
88         _bt_pos++;
89         _bt_cmd=c;
90     } else if (_bt_pos==2) {
91         _bt_pos++;
92         _bt_length=c;
93     } else if ((_bt_pos>=3)&&(_bt_pos<=( _bt_length+2))) {
94         _bt_data[_bt_pos-3]=c;
95         _bt_pos++;
96     } else if ((_bt_pos==( _bt_length+3))&&(c=='*')){
97 #ifndef PROTOCOL_DEBUG
98     Serial.println(".");
99 #endif
100     if (_bt_cmd==CMD_BOOLEAN_VAR_READ_REQ){ //Boolean read request
101         int index = _bt_data[0];
102         int value = (int)boolVars[index];
103 #ifndef PROTOCOL_DEBUG
104         if (index==VAR_TRIGGER_SWITCH) {
105             Serial.print(value?1:0);
106             Serial.print(" ");
107             Serial.println("Trigger switch read requested");
108         }
109 #endif
110         //Boolean read response
111         _bt_device.write('@');
112         _bt_device.write((byte)CMD_BOOLEAN_VAR_READ_RESP);
113         _bt_device.write((byte)2);
114         _bt_device.write((byte)index);
115         _bt_device.write((byte)(value?1:0));
116         _bt_device.write('*');
117 #ifndef PROTOCOL_DEBUG
118         Serial.print((byte) '@'); Serial.print(" ");
119         Serial.print((byte)CMD_BOOLEAN_VAR_READ_RESP); Serial.print(" ");
120         Serial.print((byte)2); Serial.print(" ");
121         Serial.print((byte)index); Serial.print(" ");
122         Serial.print((byte)((boolVars[index])?1:0)); Serial.print(" ");
123         Serial.println((byte) '*');
124 #endif
125     }
126     else if (_bt_cmd==CMD_BOOLEAN_VAR_WRITE_REQ){ //Boolean write request
127         int index = _bt_data[0];
128         bool value=(bool) (_bt_data[1]==1?true:false);
129         boolVars[index]=value;
130 #ifndef PROTOCOL_DEBUG
131         if (index==VAR_TRIGGER_SWITCH){
132             Serial.print(value?1:0);
133             Serial.print(" ");
134             Serial.println("Trigger switch write requested");
135         }
136 #endif
137     }
138 }
```



```
137     }
138     else if (_bt_cmd==CMD_INT_VAR_READ_REQ){ //Int read request
139         int index = _bt_data[0];
140         int value = intVars[index];
141 #ifdef PROTOCOL_DEBUG
142     if (index==VAR_HEIGHT){
143         Serial.print(value);
144         Serial.print(" ");
145         Serial.println("Height read requested");
146     }
147     else if (index==VAR_GENDER){
148         Serial.print(value);
149         Serial.print(" ");
150         Serial.println("Gender read requested");
151     } else if (index==VAR_AGE) {
152         Serial.print(value);
153         Serial.print(" ");
154         Serial.println("Age read requested");
155     } else if (index==VAR_VTIDAL){
156         Serial.print(value);
157         Serial.print(" ");
158         Serial.println("VTidal read requested");
159     } else if (index==VAR_TRIGGER){
160         Serial.print(value);
161         Serial.print(" ");
162         Serial.println("Trigger read requested");
163     } else if (index==VAR_RPM){
164         Serial.print(value);
165         Serial.print(" ");
166         Serial.println("RPM read requested");
167     }
168 #endif
169     //Int read response
170     _bt_device.write('@');
171     _bt_device.write((byte)CMD_INT_VAR_READ_RESP);
172     _bt_device.write((byte)3);
173     _bt_device.write((byte)index);
174     _bt_device.write((byte)((value&0xFF00)>>8));
175     _bt_device.write((byte)(value&0x00FF));
176     _bt_device.write('*');
177 #ifdef PROTOCOL_DEBUG
178     Serial.print((byte) '@'); Serial.print(" ");
179     Serial.print((byte)CMD_INT_VAR_READ_RESP); Serial.print(" ");
180     Serial.print((byte)3); Serial.print(" ");
181     Serial.print((byte)index); Serial.print(" ");
182     Serial.print((byte)((value&0xFF00)>>8)); Serial.print(" ");
183     Serial.print((byte)(value&0x00FF)); Serial.print(" ");
184     Serial.println((byte) '*');
185 #endif
186     }
187     else if (_bt_cmd==CMD_INT_VAR_WRITE_REQ){ //Int write request
188         int index = _bt_data[0];
189         int value = (int)(((int)_bt_data[1]<<8)&0xFF00)|(((int)_bt_data[2])&0x00FF));
190         intVars[index]=value;
191 #ifdef PROTOCOL_DEBUG
192     if (index==VAR_HEIGHT){
193         Serial.print(value);
194         Serial.print(" ");
195         Serial.println("Height write requested");
196     } else if (index==VAR_GENDER){
197         Serial.print(value);
198         Serial.print(" ");
199         Serial.println("Gender write requested");
200     } else if (index==VAR_AGE){
201         Serial.print(value);
202         Serial.print(" ");
203         Serial.println("Age write requested");
204     } else if (index==VAR_VTIDAL){
```



```
205     Serial.print(value);
206     Serial.print(" ");
207     Serial.println("VTidal write requested");
208 } else if (index==VAR_TRIGGER){
209     Serial.print(value);
210     Serial.print(" ");
211     Serial.println("Trigger write requested");
212 } else if (index==VAR_RPM){
213     Serial.print(value);
214     Serial.print(" ");
215     Serial.println("RPM write requested");
216 }
217 #endif
218 }
219 else if (_bt_cmd==CMD_FLOAT_VAR_READ_REQ){ //Float read request
220     int index = _bt_data[0];
221     float value = floatVars[index];
222     int* valuePtr = (int*)&value;
223     //Float read response
224     _bt_device.write('@');
225     _bt_device.write((byte)CMD_FLOAT_VAR_READ_RESP);
226     _bt_device.write((byte)5);
227     _bt_device.write((byte)index);
228     _bt_device.write((byte)((*valuePtr&0xFF000000)>>24));
229     _bt_device.write((byte)((*valuePtr&0x00FF0000)>>16));
230     _bt_device.write((byte)((*valuePtr&0x0000FF00)>>8));
231     _bt_device.write((byte)(*valuePtr&0x000000FF));
232     _bt_device.write('*');
233 #ifdef PROTOCOL_DEBUG
234     Serial.print((byte) '@'); Serial.print(" ");
235     Serial.print((byte)CMD_FLOAT_VAR_READ_RESP); Serial.print(" ");
236     Serial.print((byte)5); Serial.print(" ");
237     Serial.print((byte)index); Serial.print(" ");
238     Serial.print((byte)((*valuePtr&0xFF000000)>>24)); Serial.print(" ");
239     Serial.print((byte)((*valuePtr&0x00FF0000)>>16)); Serial.print(" ");
240     Serial.print((byte)((*valuePtr&0x0000FF00)>>8)); Serial.print(" ");
241     Serial.print((byte)(*valuePtr&0x000000FF)); Serial.print(" ");
242     Serial.println((byte) '*');
243 #endif
244 }
245 else if (_bt_cmd==CMD_FLOAT_VAR_WRITE_REQ){ //Float write request
246     int index = _bt_data[0];
247     int valueH=((int)_bt_data[1]<<8)&0xFF00|((int)_bt_data[1]);
248     int valueL=((int)_bt_data[2]<<8)&0xFF00|((int)_bt_data[3]);
249     int value=((int)((valueH<<16)&0xFFFF0000)|((int)valueL&0x0000FFFF);
250     float *valuePtr = (float*)&value;
251     floatVars[index]=*valuePtr;
252 #ifdef PROTOCOL_DEBUG
253     Serial.print(*valuePtr);
254     Serial.println(" ");
255 #endif
256 }
257 else if (_bt_cmd==CMD_STRING_VAR_READ_REQ){ //String read request
258     int index = _bt_data[0];
259     char* str = stringVars[index];
260     String Str(str);
261 #ifdef PROTOCOL_DEBUG
262     if (index==VAR_NAME){
263         Serial.print(Str);
264         Serial.print(" ");
265         Serial.println("name read requested");
266     }
267 #endif
268     int len = Str.length();
269     _bt_device.write('@');
270     _bt_device.write((byte)CMD_STRING_VAR_READ_RESP);
271     _bt_device.write((byte)2+len);
272     _bt_device.write((byte)index);
```



```
273     _bt_device.write((byte)len);
274     for (int i=0;i<len;i++)
275         _bt_device.write(Str.charAt(i));
276     _bt_device.write('*');
277 #ifdef PROTOCOL_DEBUG
278     Serial.print((byte) '@'); Serial.print(" ");
279     Serial.print((byte) CMD_STRING_VAR_READ_RESP); Serial.print(" ");
280     Serial.print((byte) 2+len); Serial.print(" ");
281     Serial.print((byte) index); Serial.print(" ");
282     Serial.print((byte) len); Serial.print(" ");
283     for (int i=0;i<len;i++){
284         Serial.print(str.charAt(i)); Serial.print(" ");}
285     Serial.println((byte) '*');
286 #endif
287     }
288     else if (_bt_cmd==CMD_STRING_VAR_WRITE_REQ){ //String write request
289         int index = _bt_data[0];
290         char *str = stringVars[index];
291         for (int i=0;i<_bt_length;i++)
292             str[i]=(char)_bt_data[i+1];
293     }
294     _bt_pos=0;
295     _bt_length=0;
296 } else{
297     _bt_pos=0;
298     _bt_length=0;
299 }
300 }
```



REFERENCE MANUAL

Components

The extension has the following components, which are classified according to the colour code indicated in the following figure. To add the components, drag them to the interface in the designer view and they will appear as non-visible components.

The main component is the **FacilinoBluetoothClient** component that allows communication with the Bluetooth device. This component allows you to connect / disconnect with the device, as well as different utilities to reconnect with the last device and know if it is paired. It has a timer that allows the regularity with which the availability of new data is checked (reception).

On the other hand, the components of digital, analog signals, sensors and actuators and variables all work in a very similar way. Those components that read information have the "Request" and "Update" methods, whose functionality is similar, unlike "Request" is non-blocking, while "Update" is blocking. After making a call to any of the methods, the component will report with the information received through the "Received" event. In addition, if for any reason the information is not received, this will be notified through the "Timeout" event.

—	Bluetooth Client
—	Analog/Digital signals
—	Sensors/Actuators
—	Variables

- AnalogReadBluetooth
- AnalogWriteBluetooth
- BooleanVariableBluetooth
- BuzzerBluetooth
- DHTBluetooth
- DigitalReadBluetooth
- DigitalWriteBluetooth
- FacilinoBluetoothClient
- FloatVariableBluetooth
- IntVariableBluetooth
- LEDMatrix8x8Bluetooth
- RGB_LEDStripBluetooth
- ServoBluetooth
- ServoContBluetooth
- SonarBluetooth
- StringVariableBluetooth

FacilinoBluetoothClient

Properties

AdressesAndNames *list read-only*

List with addresses and names of paired Bluetooth devices.

Available *boolean read-only*

Returns true if Bluetooth is available on the device.

Enabled *boolean read-only*

Returns true if Bluetooth is enabled on the device.



Connected *boolean read-only*

Returns true if connected to a Bluetooth device.

LastConnectedAddress *text read-only*

Gets the address and name of the last connected (and saved) device.

TimerEnabled *boolean*

Gets or sets the timer to notify the reception of new data.

TimerInterval *number*

Gets or sets the interval in milliseconds of the timer to notify the reception of new data.

FacilinoBluetoothClient *component*

A FacilinoBluetoothClient component.

Methods

boolean Connect (*text* address)

To connect to a device, we will use the “Connect” method that requires us to pass it a MAC address of the bluetooth device provided by the “AddressesAndNames” property. Returns true or false depending on whether the connection was established.

Disconnect

Disconnect the current connection.

ForgetLastConnection

Allows you to forget the last connection established (and saved).

boolean IsDevicePaired (*text* address)

Returns true if the device indicated by the MAC address is paired. We can only connect to paired devices before trying to reconnect. Therefore, we should check that the last address with which the connection was established is still paired.

Reconnect

Allows to reconnect with the last connection established (and saved).

SaveConnection

Saves address and name of the last connection established, to be able to reconnect easily.

Events

TelegramError Notifies that an error has been generated in the telegram.

error *text* Telegram error message.

TelegramReceived Notifica que se ha recibido un telegrama.

cmd *number* Command of the received telegram.

length *number* Number of bytes received in the data.

data *list* List with telegram data.

DigitalReadBluetooth

Properties

FacilinoBluetoothClient *component*



Gets or sets the component that manages Bluetooth communication. Must be a component of type "FacilinoBluetoothClient".

Pin *number*

Gets or sets the digital input pin number.

UpdateTimeout *number*

Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *boolean*

Gets the value of the digital input value (last received value).

DigitalReadBluetooth *component*

The DigitalReadBluetooth component.

Methods

Request

Request the reading of the digital input with the indicated pin number (sends a reading telegram with the pin number). This feature is non-blocking and the reception of the telegram is expected to be notified with the response with "Received" or "Changed" events.

Update

Request the reading of the pin number (sends a read telegram with the pin number and waits to receive the telegram with the information of the variable with the same pin number). This is a blocking method. "Timeout" property indicates the maximum time this function is blocking before "Timeout" event is generated.

Events

Changed

Notifies that the digital input value has changed (a telegram has been received indicating that change).

value *boolean*

Value of the digital input.

Received

Notifies that a telegram has been received with information on the value of the digital input.

value *boolean*

Value of the digital input.

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.

DigitalWriteBluetooth

Properties

FacilinoBluetoothClient *component*



Gets or sets the component that manages Bluetooth communication. Must be a component of type “FacilinoBluetoothClient”.

Pin *number*

Gets or sets the digital output pin number.

Value *boolean*

Gets the value of the last value set.

DigitalWriteBluetooth *component*

The digitalWriteBluetooth component.

Methods

Set (*boolean* value)

Sets the value of the digital output (sends a write telegram with the pin number).

Toggle

Toggles the value of the digital output (sends a write telegram with the opposite value to the last value stored with the pin number).

Events

None

AnalogReadBluetooth

Properties

FacilinoBluetoothClient *component*

Gets or sets the component that manages Bluetooth communication. Must be a component of type “FacilinoBluetoothClient”.

Pin *number*

Gets or sets the analog input pin number.

UpdateTimeout *number*

Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *number*

Gets the value of the analog input value (last received value).

AnalogReadBluetooth *component*

The AnalogReadBluetooth component.

Methods

Request

Request the reading of the analog input with the indicated analog pin number (sends a reading telegram with the analog pin number). This feature is non-blocking and the reception of the telegram is expected to be notified with the response with “Received” event.

Update

Request the reading of the analog pin number (sends a read telegram with the pin number and waits to receive the telegram with the information of the variable with the same pin number). This is a blocking



method. “Timeout” property indicates the maximum time this function is blocking before “Timeout” event is generated.

Events

Received

Notifies that a telegram has been received with information on the value of the analog input.

value *number*

Value of the analog input.

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.

AnalogWriteBluetooth

Properties

FacilinoBluetoothClient *component*

Gets or sets the component that manages Bluetooth communication. Must be a component of type “FacilinoBluetoothClient”.

Pin *number*

Gets or sets the PWM digital output pin number.

Value *number*

Gets the value of the last value set.

DigitalWriteBluetooth *component*

The AnalogWriteBluetooth component.

Methods

Set (*number* value)

Sets the value of the PWM digital output (sends a write telegram with the pin number). A value between 0 and 65535 (16 bit number).

Events

None

BooleanVariableBluetooth

Properties

FacilinoBluetoothClient *component*

Gets or sets the component that manages Bluetooth communication. Must be a component of type “FacilinoBluetoothClient”.

Index *number*

Gets or sets the index of the boolean variable position within the list of boolean variables that the device with which we communicate will maintain.

UpdateTimeout *number*



Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *boolean*

Gets the value of the boolean variable (last received value).

BooleanVariableBluetooth *component*

The BooleanVariableBluetooth component.

Methods

Request

Request the reading of the boolean variable with the indicated index (sends a reading telegram with the variable index). This feature is non-blocking and the reception of the telegram is expected to be notified with the response with "Received" or "Changed" events.

Set (*boolean* value)

Sets the value of the boolean variable (sends a write telegram with the variable index).

Toggle

Toggles the value of the Boolean variable (sends a write telegram with the opposite value to the last value stored with the variable index).

Update

Request the reading of the boolean variable (sends a read telegram with the variable index and waits to receive the telegram with the information of the variable with the same index). This is a blocking method. "Timeout" property indicates the maximum time this function is blocking before "Timeout" event is generated.

Events

Changed

Notifies that the boolean variable has changed (a telegram has been received indicating that change).

value *boolean*

Value of the boolean variable.

Received

Notifies that a telegram has been received with information on the value of the boolean variable.

value *boolean*

Value of the boolean variable.

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.

IntVariableBluetooth

Properties

FacilinoBluetoothClient *component*



Gets or sets the component that manages Bluetooth communication. Must be a component of type "FacilinoBluetoothClient".

Index *number*

Gets or sets the index of the integer variable position within the list of integer variables that the device with which we communicate will maintain.

UpdateTimeout *number*

Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *number*

Gets the value of the integer variable (last received value). The number is between 0 and 65535 (a 16-bit number).

IntVariableBluetooth *component*

The IntVariableBluetooth component.

Methods

Request

Request the reading of the integer variable with the indicated index (send a reading telegram with the variable index). This feature is non-blocking and is expected to be notified the reception of the telegram with the response with "Received" event.

Set (*number* value)

Sets the value of the integer variable (sends a write telegram with the variable index). The value must be between 0 and 65535 (16-bit number).

Update

Request the reading of the integer variable (sends a read telegram with the variable index and waits to receive the telegram with the information of the variable with the same index). This is a blocking method. "Timeout" property indicates the maximum time this function is blocking before "Timeout" event is generated.

Events

Received

Notifies that a telegram has been received with information on the value of the integer variable.

value *number*

Value with the integer variable. A number between 0 and 65535 (a 16-bit number).

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.

FloatVariableBluetooth

Properties

FacilinoBluetoothClient *component*

Gets or sets the component that manages Bluetooth communication. Must be a component of type "FacilinoBluetoothClient".



Index *number*

Gets or sets the index of the float variable position within the list of float variables that the device with which we communicate will maintain.

UpdateTimeout *number*

Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *number*

Gets the value of the float variable (last received value). The number is 32-bit precision.

FloatVariableBluetooth *component*

The FloatVariableBluetooth component.

Methods

Request

Request the reading of the float variable with the indicated index (send a reading telegram with the variable index). This feature is non-blocking and the reception of the telegram is expected to be notified with the response with "Received" event.

Set (*number* value)

Sets the value of the float variable (sends a write telegram with the variable index). The number sent is 32-bit precision.

Update

Request the reading of the float variable (sends a read telegram with the variable index and waits to receive the telegram with the information of the variable with the same index). This is a blocking method. "Timeout" property indicates the maximum time this function is blocking before "Timeout" event is generated.

Events

Received

Notifies that a telegram has been received with information on the value of the float variable.

value *number*

Value with the float variable. The number received is 32-bit precision.

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.

StringVariableBluetooth

Properties

FacilinoBluetoothClient *component*

Gets or sets the component that manages Bluetooth communication. Must be a component of type "FacilinoBluetoothClient".

Index *number*



Gets or sets the index of the String variable position within the list of String variables that the device with which we communicate will maintain.

UpdateTimeout *number*

Gets or sets the maximum time that must elapse to consider that an error has occurred in the reception of the telegram.

Value *text*

Gets the value of the String variable (last received value). Maximum amount of string characters is 252.

StringVariableBluetooth *component*

The StringVariableBluetooth component.

Methods

Request

Request the reading of the String variable with the indicated index (send a reading telegram with the variable index). This feature is non-blocking and the reception of the telegram is expected to be notified with the response with "Received" event.

Set (*text value*)

Sets the value of the String variable (sends a write telegram with the variable index). The maximum number of string characters is 252.

Update

Request the reading of the String variable (sends a read telegram with the variable index and waits to receive the telegram with the information of the variable with the same index). This is a blocking method. "Timeout" property indicates the maximum time this function is blocking before "Timeout" event is generated.

Events

Received

Notifies that a telegram has been received with information on the value of the float variable.

value *text*

Value with the String variable. The maximum number of received characters is 252.

Timeout

Notifies that a read timer overflow has occurred.

error *text*

Text with the timeout information.



PROTOCOL

Telegram structure

Every transmitted telegram has the following structure

STX	CMD	LENGTH	DATA₀	...	DATA_{N-1}	END
------------	------------	---------------	-------------------------	-----	---------------------------	------------

STX (1 byte): '@' symbol.

CMD (1 byte): See command list.

LENGTH (1 byte): Number of data length (see command list)

DATA (N bytes): Data (see command list).

END (1 byte): '*' symbol.

Command List

The following list of commands describes all possible telegrams to be implemented.

Digital Read Request (App → Arduino)

Request a digital input read for the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	END
@	0x00	1	Pin	*

Digital Read Response (Arduino → App)

Response to a digital read input request for the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x01	2	Pin	Value	*

Digital Write (App → Arduino)

Sets the value of a digital output for the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x02	2	Pin	Value	*

Analog Read Request (App → Arduino)

Request an analog input read for the indicated analog pin.

STX	CMD	LENGTH	DATA ₀	END
@	0x03	1	Pin	*

Analog Read Response (Arduino → App)

Response to an analog read input request for the indicated analog pin. Value is between 0 to 65535 (16-bit value).



STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	END
@	0x04	3	Pin	Value	*

Analog Write (App → Arduino)

Sets the value (0-65535, 16-bit value) for an analog output (PWM digital output) for the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁	DATA ₂	END
@	0x05	3	Pin	Value		*

Servo (App → Arduino)

Sets the angle (0° to 180°, 8-bit data) of a conventional servo connected to the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x10	2	Pin	Angle	*

Continuous Servo (App → Arduino)

Sets the velocity (-100% to 100%, 8-bit data) of a continuous rotation servo connected to the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x11	2	Pin	Velocity	*

Sonar Request (App → Arduino)

Request a distance measurement from a sonar sensor connected to the indicated digital pins.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x12	2	Echo pin	Trigger Pin	*

Sonar Response (Arduino → App)

Response to a distance measurement (0-65536, 16-bit value) of a sonar sensor connected to the indicated digital pins.

STX	CMD	LENGTH	DATA ₀	DATA ₁	DATA ₂₋₃	END
@	0x13	4	Echo pin	Trigger Pin	Distance	*

Buzzer Tone (App → Arduino)

Reproduce a tone in a piezo-electric buzzer connected to the indicated digital pin. The buzzer will vibrate at the given frequency (in Hz, 16-bit value) and duration (in ms, 16-bit value).

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	DATA ₃₋₄	END
@	0x20	2	Pin	Frequency	Duration	*



Buzzer Melody (App → Arduino)

Reproduces a melody (a list of consecutive tones with frequencies and durations {Freq.₀, Dur.₀, Freq.₁, Dur.₁, ..., Freq._M, Dur._M}) in a piezo-electric buzzer connected to the indicated digital pin. The buzzer will vibrate at the given frequencies (in Hz, 16-bit value) and durations (in ms, 16-bit value).

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	DATA ₃₋₄	...	DATA _{N-4-N-2}	DATA _{N-2-N-1}	END
@	0x21	4*M+1	Pin	Freq. ₀	Dur. ₀		Freq. _{M-1}	Dur. _{M-1}	*

DHT Request (App → Arduino)

Request a temperature and humidity measurements from a digital DHT sensor connected to the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	END
@	0x22	2	Pin	*

DHT Response (Arduino → App)

Response to a temperature (Celsius, 16-bit value) and humidity (relative humidity in %, 16-bit value) measurement of a digital DHT sensor connected to the indicated digital pin.

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	DATA ₃₋₄	END
@	0x23	2	Pin	Temperature	Humidity	*

8x8 LED Matrix (App → Arduino)

Sets the LEDs of an 8x8 LED Matrix connected to indicated pins with the given LED values. LEDs of each column of the 8x8 LED Matrix will be controlled with the corresponding column value with the corresponding decoded binary value (8-bit, value of each column).

STX	CMD	LENGTH	DATA ₀	DATA ₁	DATA ₂	DATA ₃	...	DATA ₁₀	END
@	0x50	11	CLK Pin	DIN Pin	CS Pin	Col ₁		Col ₈	*

8x8 LED Matrix Predefined Expression (App → Arduino)

Sets the LEDs of a 8x8 LED Matrix connected to indicated pins. The expression is a number that will be used to show a predefined set of expressions (this depends on the Arduino side).

STX	CMD	LENGTH	DATA ₀	DATA ₁	DATA ₂	DATA ₃	END
@	0x51	4	CLK Pin	DIN Pin	CS Pin	Expression	*

RGB LED Strip (App → Arduino)

Sets the LEDs of a RGB LED Strip connected to indicated digital pin with the given sequence of RGB colours {RGB₀, RGB₁, ..., RGB_{M-1}}. Each colour is a 24-bits value for red (8-bit), green (8-bit) and blue (8-bit) channels.

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₃	...	DATA ₁₀	END
@	0x60	3*M+1	Pin	RGB ₀		RGB _{M-1}	*

RGB LED Strip Predefined Expression (App → Arduino)



Sets the LEDs of a RGB LED Strip connected to indicated digital pin. The expression or sequence is a number that will be used to show a predefined set of LED combination (this depends on the Arduino side).

STX	CMD	LENGTH	DATA ₀	DATA ₃	END
@	0x61	2	Pin	Expression	*

Sets Boolean Variable (App → Arduino)

Sets the value of a boolean variable (1->>true, 0->>false) at the indicated position (index). Arduino will keep a list of boolean variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x80	2	Index	Value	*

Request Boolean Variable (App → Arduino)

Requests the value of a boolean variable (1->>true, 0->>false) at the indicated position (index). Arduino will keep a list of boolean variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	END
@	0x81	1	Index	*

Response Boolean Variable Request (Arduino → App)

Requests the value of a boolean variable (1->>true, 0->>false) at the indicated position (index). Arduino will keep a list of boolean variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA ₁	END
@	0x82	2	Index	Value	*

Sets Integer Variable (App → Arduino)

Sets the value of an integer variable (16-bit value) at the indicated position (index). Arduino will keep a list of integer variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	END
@	0x83	3	Index	Value	*

Request Integer Variable (App → Arduino)

Requests the value of an integer variable (16-bit value) at the indicated position (index). Arduino will keep a list of integer variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	END
@	0x84	1	Index	*

Response Integer Variable Request (Arduino → App)

Requests the value of an integer variable (16-bit value) at the indicated position (index). Arduino will keep a list of integer variables to communicate with the App.



STX	CMD	LENGTH	DATA ₀	DATA ₁₋₂	END
@	0x85	3	Index	Value	*

Sets Float Variable (App→Arduino)

Sets the value of a float variable (32-bit value) at the indicated position (index). Arduino will keep a list of float variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₄	END
@	0x86	5	Index	Value	*

Request Float Variable (App→Arduino)

Requests the value of a float variable (32-bit value) at the indicated position (index). Arduino will keep a list of float variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	END
@	0x87	1	Index	*

Response Float Variable Request (Arduino→App)

Requests the value of a float variable (32-bit value) at the indicated position (index). Arduino will keep a list of float variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA ₁₋₄	END
@	0x88	5	Index	Value	*

Sets String Variable (App→Arduino)

Sets the value of a String variable (up-to 252 characters) at the indicated position (index). Arduino will keep a list of String variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA _{1-M}	END
@	0x89	M+1	Index	String	*

Request String Variable (App→Arduino)

Requests the value of a String variable (up-to 252 characters) at the indicated position (index). Arduino will keep a list of String variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	END
@	0x8A	1	Index	*

Response String Variable Request (Arduino→App)

Requests the value of a String variable (up-to 252 characters) at the indicated position (index). Arduino will keep a list of String variables to communicate with the App.

STX	CMD	LENGTH	DATA ₀	DATA _{1-M}	END
@	0x8B	M+1	Index	String	*