# 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).

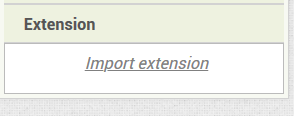
**DOWNLOAD**

**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**

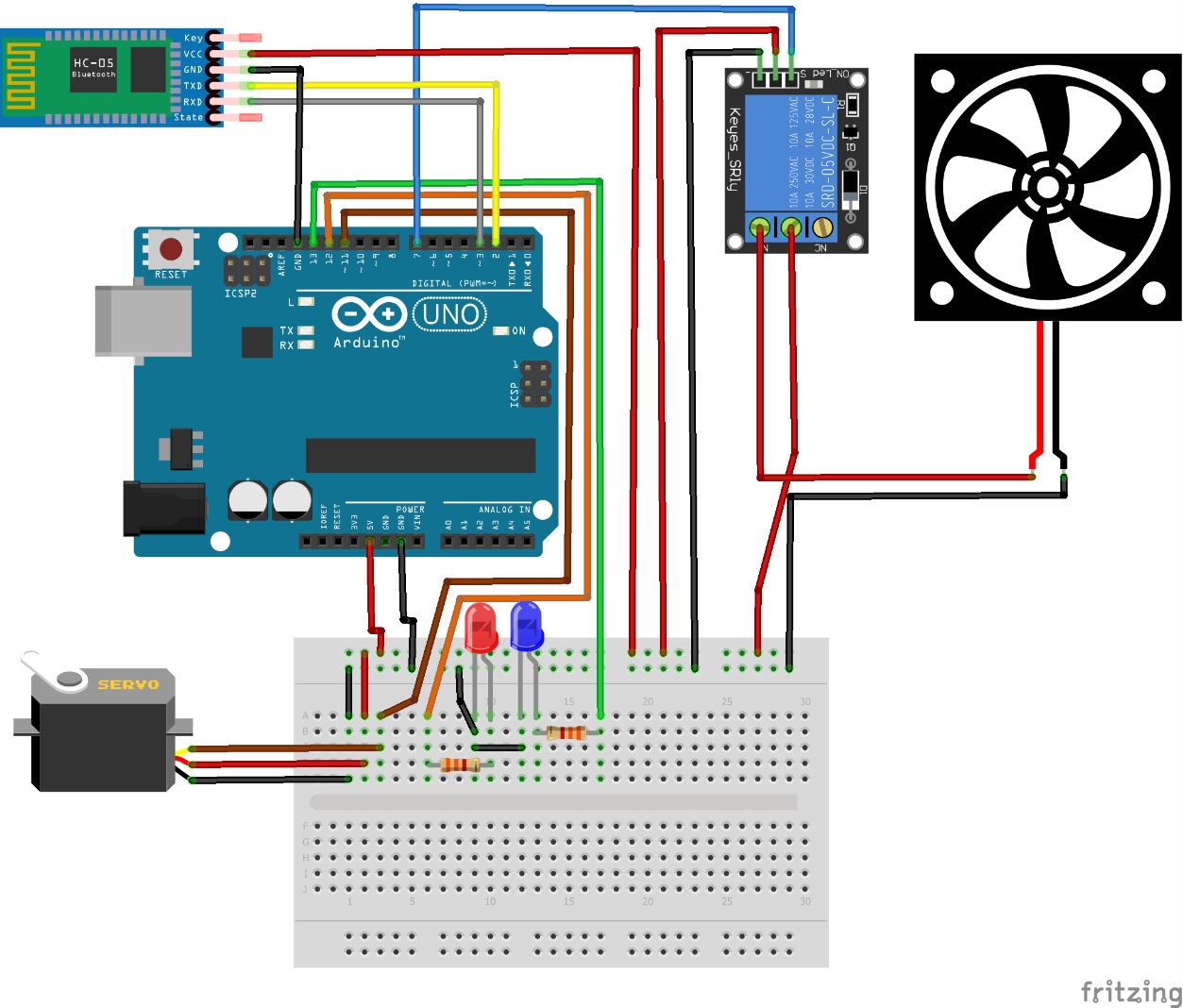
**Download**

<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:**

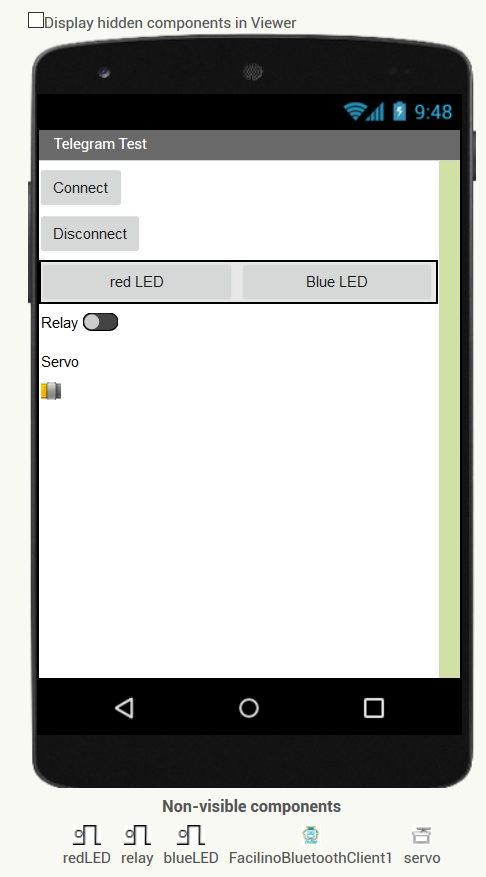
* **Red LED pin 12.**
* **Blue LED pin 13.**
* **Relay module pin 7.**
* **Servo pin 11.**
* **Bluetooth module connected to pin D2 (TX) and pin D3 (RX).**

****

**App Inventor**

**Designer instructions**

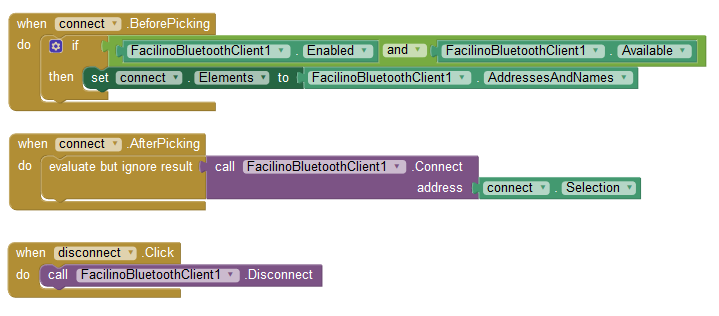
* **Create a ListPicker component and rename it as *‘connect’*.**
* **Create a Button component and rename it as *‘disconnect’*.**
* **Create two Buttons components inside a HorizontalArragement and rename them as *‘butRed’* and ‘*butBlue’*.**
* **Create Switch component and rename it as *‘relaySwitch’*.**
* **Create a Slider component and rename it as *‘servoSlider’*.**
* **Import the FacilinoBluetooth AI2 extension (es.roboticafacil.facilino.runtime.bluetooth.aix).**
* **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.**

****

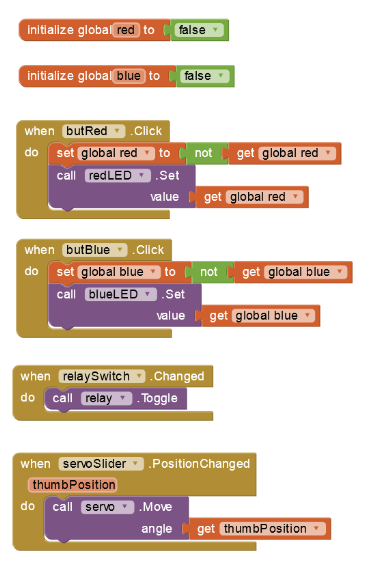
**Blocks**

**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.**

****

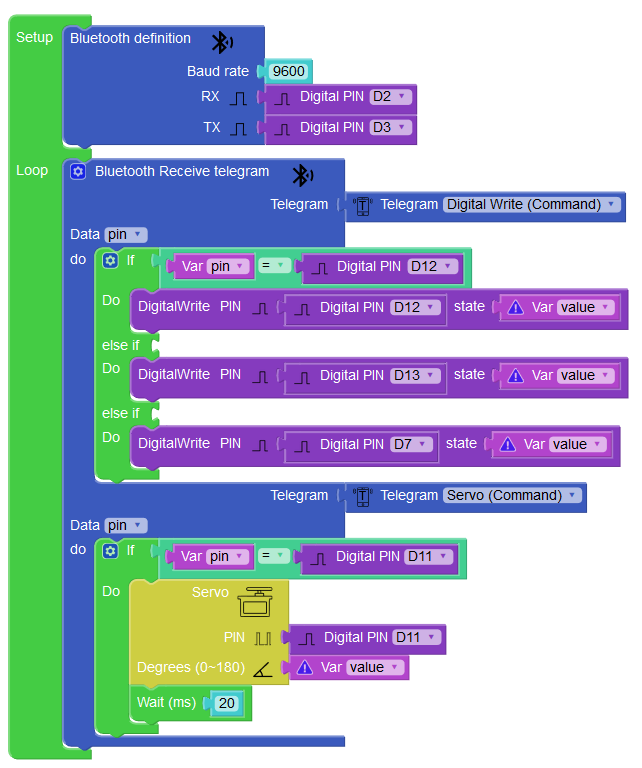
**To set digital outputs, we can use either the ‘*Set’* method or the ‘*Toggle’* method. To set the position of the servo, we can use the ‘*Move’* method.**

****

**Facilino Code**

**The *Bluetooth definition* instruction uses pins D2 and D3, to stablish 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.**

**#include <SoftwareSerial.h>**

**#include <Servo.h>**

**#define BT\_TX\_PIN 2 //TX pin of bluetooth module**

**#define BT\_RX\_PIN 3 //RX pin of bluetooth module**

**#define RED\_LED\_PIN 12**

**#define BLUE\_LED\_PIN 13**

**#define RELAY\_PIN 7**

**#define SERVO\_PIN 11**

**SoftwareSerial \_bt\_device(BT\_TX\_PIN,BT\_RX\_PIN);**

**int \_bt\_pos=0;**

**unsigned char \_bt\_cmd=0;**

**int \_bt\_length=0;**

**unsigned char \_bt\_data[255];**

**#define CMD\_DIGITAL\_WRITE 0x02**

**#define CMD\_SERVO 0x10**

**Servo \_servo;**

**void setup()**

**{**

**\_bt\_device.begin(9600);**

**\_bt\_device.flush();**

**pinMode(RED\_LED\_PIN,OUTPUT);**

**pinMode(BLUE\_LED\_PIN,OUTPUT);**

**pinMode(RELAY\_PIN,OUTPUT);**

**\_servo.attach(SERVO\_PIN);**

**}**

**void loop()**

**{**

**if (\_bt\_device.available()>0)**

**{**

**unsigned char c;**

**\_bt\_device.readBytes(&c,1);**

**if ((c=='@')&&(\_bt\_pos==0))**

**\_bt\_pos++;**

**else if (\_bt\_pos==1){**

**\_bt\_pos++;**

**\_bt\_cmd=c;**

**}**

**else if (\_bt\_pos==2){**

**\_bt\_pos++;**

**\_bt\_length=c;**

**}**

**else if ((\_bt\_pos>=3)&&(\_bt\_pos<=(\_bt\_length+2))){**

**\_bt\_data[\_bt\_pos-3]=c;**

**\_bt\_pos++;**

**}**

**else if ((\_bt\_pos==(\_bt\_length+3))&&(c=='\*')){**

**if (\_bt\_cmd==CMD\_DIGITAL\_WRITE){**

**int pin = \_bt\_data[0];**

**boolean value = \_bt\_data[1]==1? HIGH: LOW;**

**if (pin==RED\_LED\_PIN)**

**digitalWrite(RED\_LED\_PIN,value);**

**else if (pin==BLUE\_LED\_PIN)**

**digitalWrite(BLUE\_LED\_PIN,value);**

**else if (pin==RELAY\_PIN)**

**digitalWrite(RELAY\_PIN,value);**

**}**

**if (\_bt\_cmd==CMD\_SERVO){**

**int pin = \_bt\_data[0];**

**byte value = \_bt\_data[1];**

**if (pin==SERVO\_PIN) {**

**\_servo.write(value);**

**delay(20);**

**}**

**}**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**else{**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**}**

## **Example 2: Multisensor Shield with Arduino Uno**

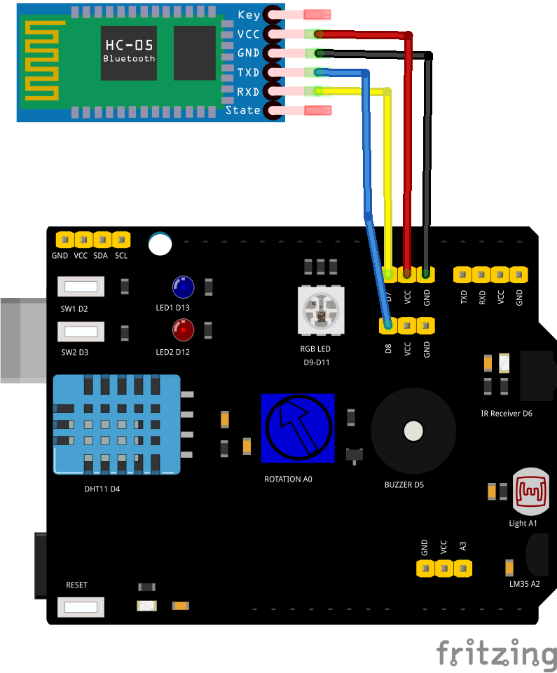
**Download**

<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**

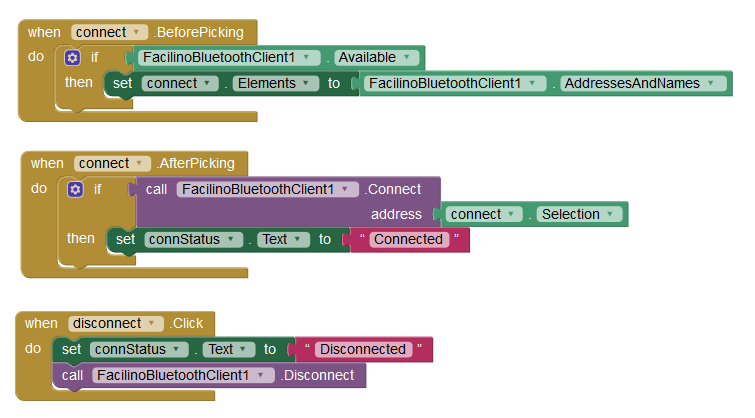
* **Create a ListPicker component and rename it as *‘connect’*, create a Button component and rename it as *‘disconnect’* and create a Label component an rename it as ‘*connStatus’*. All these components must be horizontally arranged with a HorizontalArragement component. Make sure that the HorizontalArragement, 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 HorizontalArragement 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 HorizontalArragement compoments fill the parent width (in the ‘*Width’* property). Modify the ‘*Text’* property of static Label components as shown in the image.**
* **Inside a HorizontalArragement component create nested HorizontalArragement and VerticalArragement 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 VerticalArragement 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 HorizontalArragement component and a nested VerticalArragement component and a Button component. Inside the VerticalArregement component create two HorizontalArragement 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 HorizontalArragement 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 ‘*TimerInverval’* property to 100 (milliseconds) for ‘Clock1’ and 2000 for ‘Clock2’.**
* **Import the FacilinoBluetooth AI2 extension (es.roboticafacil.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.**

****

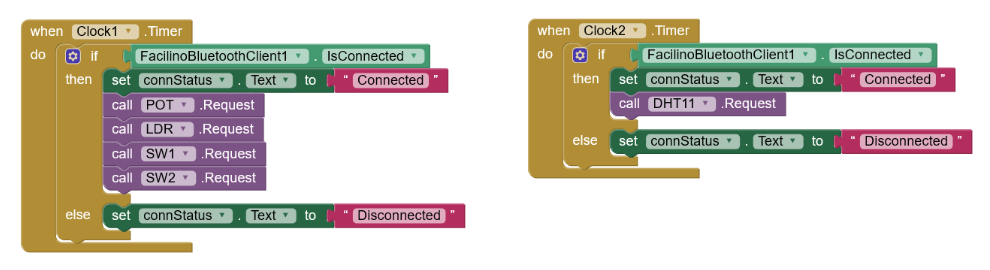
**Blocks**

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

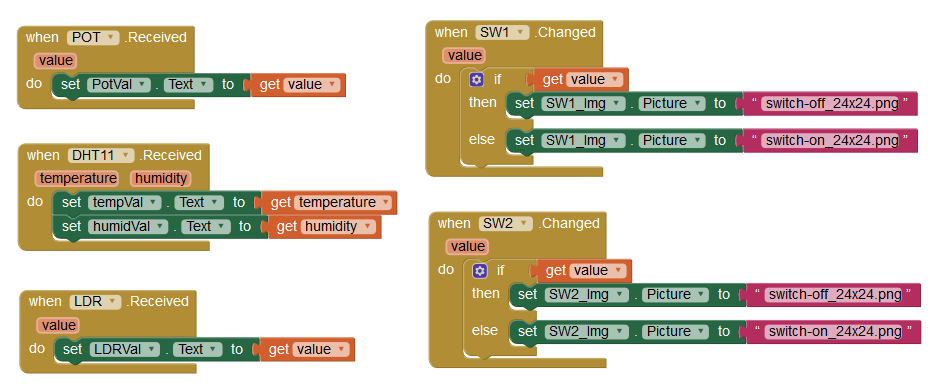
**Connection and disconnection use the ‘*Connect’* and ‘*Disconnect’* methods of the FacilinoBluetoothClient component. To list the MAC address in the ListPicker component, use the ‘AddressesAndNames’ property. To check if Bluetooth is ‘*Enabled’* and ‘*Available’*, use the corresponding properties.**

****

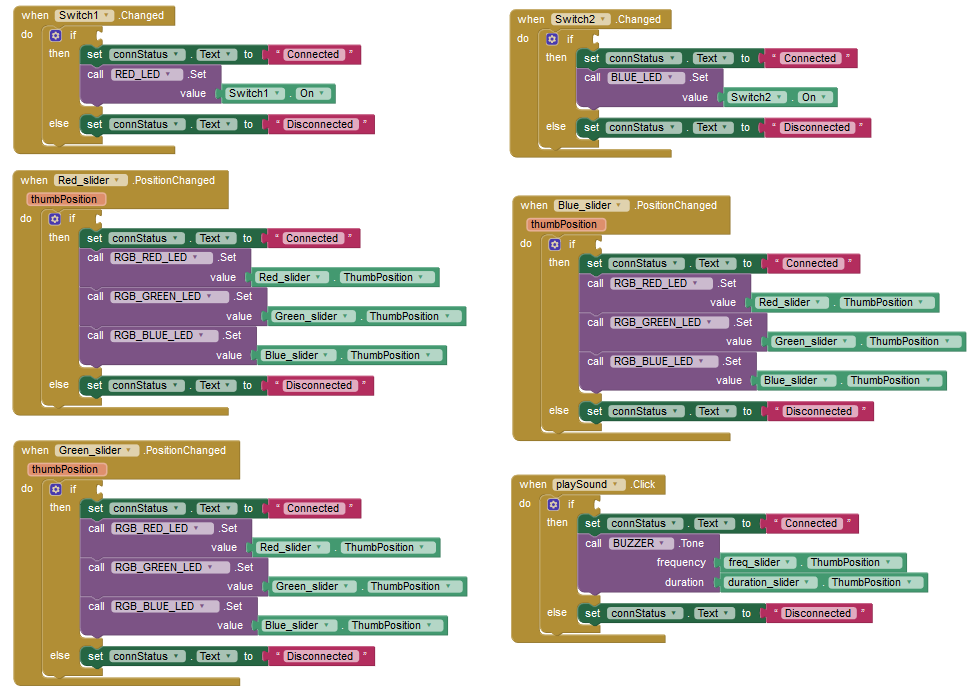
**To read from all digital and analog signals, such as the potentiometer, the LDR and the two push-buttons, we can use the ‘*Request’* method, included on each of the corresponding components. This can be done inside the ‘*Timer’* event of the *‘Clock1’* component. DHT sensor can be requested inside 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.**

****

**Upon each request, we need to handle the ‘Received’ events, if received.**

****

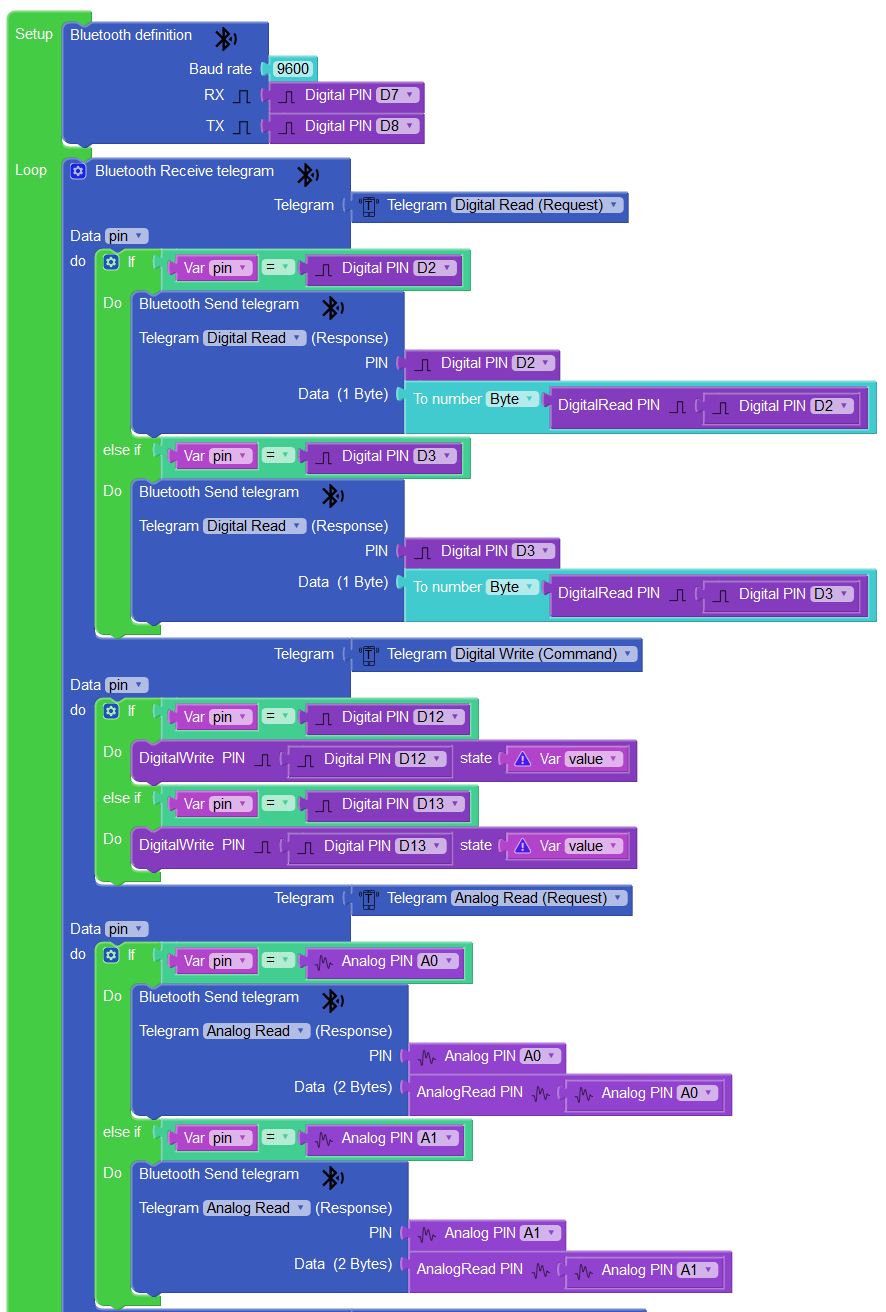
**Finally, let’s handle the digital outputs of the board, such as the red and blue LEDs, the RGB LEDs by calling the ‘Set’ method and the Buzzer component which can produce tones using the ‘Tone’ method:**

****

**Facilino Code**

**The *Bluetooth definition* instruction uses pins D7 and D8, to stablish 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.**

****

****

**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 form the telegram data such as pin and value. Depending on the receive telegram, the code between lines 71 and 156 performs the corresponding action.**

**#include <SoftwareSerial.h>**

**#include <DHT.h>**

**#define BT\_TX\_PIN 7 //TX pin of bluetooth module**

**#define BT\_RX\_PIN 8 //RX pin of bluetooth module**

**#define RED\_LED\_PIN 12**

**#define BLUE\_LED\_PIN 13**

**#define SW1\_PIN 2**

**#define SW2\_PIN 3**

**#define DHT11\_PIN 4**

**#define BUZZER\_PIN 5**

**#define RGB\_RED\_PIN 9**

**#define RGB\_GREEN\_PIN 11**

**#define RGB\_BLUE\_PIN 10**

**#define POT\_PIN A0**

**#define LDR\_PIN A1**

**SoftwareSerial \_bt\_device(BT\_TX\_PIN,BT\_RX\_PIN);**

**DHT sensorDHT11(DHT11\_PIN,DHT11);**

**int \_bt\_pos=0;**

**unsigned char \_bt\_cmd=0;**

**int \_bt\_length=0;**

**unsigned char \_bt\_data[255];**

**#define CMD\_DIGITAL\_READ\_REQ 0x00**

**#define CMD\_DIGITAL\_READ\_RESP 0x01**

**#define CMD\_DIGITAL\_WRITE 0x02**

**#define CMD\_ANALOG\_READ\_REQ 0x03**

**#define CMD\_ANALOG\_READ\_RESP 0x04**

**#define CMD\_ANALOG\_WRITE 0x05**

**#define CMD\_BUZZER\_TONE 0x20**

**#define CMD\_DHT\_REQ 0x22**

**#define CMD\_DHT\_RESP 0x23**

**void setup()**

**{**

**\_bt\_device.begin(9600);**

**\_bt\_device.flush();**

**pinMode(RED\_LED\_PIN,OUTPUT);**

**pinMode(BLUE\_LED\_PIN,OUTPUT);**

**pinMode(SW1\_PIN,OUTPUT);**

**pinMode(SW2\_PIN,OUTPUT);**

**pinMode(BUZZER\_PIN,OUTPUT);**

**pinMode(RGB\_RED\_PIN,OUTPUT);**

**pinMode(RGB\_GREEN\_PIN,OUTPUT);**

**pinMode(RGB\_BLUE\_PIN,OUTPUT);**

**sensorDHT11.begin();**

**}**

**void loop()**

**{**

**if (\_bt\_device.available()>0)**

**{**

**unsigned char c;**

**\_bt\_device.readBytes(&c,1);**

**if ((c=='@')&&(\_bt\_pos==0))**

**\_bt\_pos++;**

**else if (\_bt\_pos==1){**

**\_bt\_pos++;**

**\_bt\_cmd=c;**

**}**

**else if (\_bt\_pos==2){**

**\_bt\_pos++;**

**\_bt\_length=c;**

**}**

**else if ((\_bt\_pos>=3)&&(\_bt\_pos<=(\_bt\_length+2))){**

**\_bt\_data[\_bt\_pos-3]=c;**

**\_bt\_pos++;**

**}**

**else if ((\_bt\_pos==(\_bt\_length+3))&&(c=='\*')){**

**if (\_bt\_cmd==CMD\_DIGITAL\_READ\_REQ){**

**int pin=\_bt\_data[0];**

**if (pin==SW1\_PIN){**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_DIGITAL\_READ\_RESP);**

**\_bt\_device.write((byte)2);**

**\_bt\_device.write((byte)SW1\_PIN);**

**\_bt\_device.write((byte)(byte)(digitalRead(SW1\_PIN)));**

**\_bt\_device.write('\*');**

**}**

**else if (pin==SW2\_PIN) {**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_DIGITAL\_READ\_RESP);**

**\_bt\_device.write((byte)2);**

**\_bt\_device.write((byte)SW2\_PIN);**

**\_bt\_device.write((byte)(byte)(digitalRead(SW2\_PIN)));**

**\_bt\_device.write('\*');**

**}**

**}**

**else if (\_bt\_cmd==CMD\_DIGITAL\_WRITE){**

**int pin=\_bt\_data[0];**

**boolean value = \_bt\_data[1]==1? HIGH: LOW;**

**if (pin==RED\_LED\_PIN)**

**digitalWrite(RED\_LED\_PIN,value);**

**else if (pin==BLUE\_LED\_PIN)**

**digitalWrite(BLUE\_LED\_PIN,value);**

**}**

**else if (\_bt\_cmd==CMD\_ANALOG\_READ\_REQ){**

**int pin=\_bt\_data[0];**

**if (pin==POT\_PIN){**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_ANALOG\_READ\_RESP);**

**\_bt\_device.write((byte)3);**

**\_bt\_device.write((byte)POT\_PIN);**

**short int \_value=analogRead(POT\_PIN);**

**\_bt\_device.write((byte)((\_value&0xFF00)>>8));**

**\_bt\_device.write((byte)(\_value&0x00FF));**

**\_bt\_device.write('\*');**

**}**

**else if (pin==LDR\_PIN) {**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_ANALOG\_READ\_RESP);**

**\_bt\_device.write((byte)3);**

**\_bt\_device.write((byte)LDR\_PIN);**

**short int \_value=analogRead(LDR\_PIN);**

**\_bt\_device.write((byte)((\_value&0xFF00)>>8));**

**\_bt\_device.write((byte)(\_value&0x00FF));**

**\_bt\_device.write('\*');**

**}**

**}**

**else if (\_bt\_cmd==CMD\_ANALOG\_WRITE){**

**int pin = \_bt\_data[0];**

**byte value = \_bt\_data[1];**

**if (pin==RGB\_RED\_PIN)**

**analogWrite(RGB\_RED\_PIN,value);**

**else if (pin==RGB\_GREEN\_PIN)**

**analogWrite(RGB\_GREEN\_PIN,value);**

**else if (pin==RGB\_BLUE\_PIN)**

**analogWrite(RGB\_BLUE\_PIN,value);**

**}**

**else if (\_bt\_cmd==CMD\_DHT\_REQ){**

**int pin = \_bt\_data[0];**

**if (pin ==DHT11\_PIN) {**

**byte data[4]={0,0,0,0};**

**int temp=sensorDHT11.readTemperature();**

**int humid=sensorDHT11.readHumidity();**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_DHT\_RESP);**

**\_bt\_device.write((byte)5);**

**\_bt\_device.write((byte)DHT11\_PIN);**

**\_bt\_device.write((byte)(((temp)>>(8))));**

**\_bt\_device.write((byte)(temp));**

**\_bt\_device.write((byte)(((humid)>>(8))));**

**\_bt\_device.write((byte)(byte)(humid));**

**\_bt\_device.write('\*');**

**}**

**}**

**if (\_bt\_cmd==CMD\_BUZZER\_TONE) {**

**int pin = \_bt\_data[0];**

**int frequency = ((((int)\_bt\_data[1])<<8)|(\_bt\_data[2]));**

**int duration = ((((int)\_bt\_data[3])<<8)|(\_bt\_data[4]));**

**if (pin ==BUZZER\_PIN)**

**tone(BUZZER\_PIN,frequency,duration);**

**delay(duration);**

**noTone(BUZZER\_PIN);**

**}**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**else{**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**}**

**}**

## **Example 3: Remote Control of a Robot**

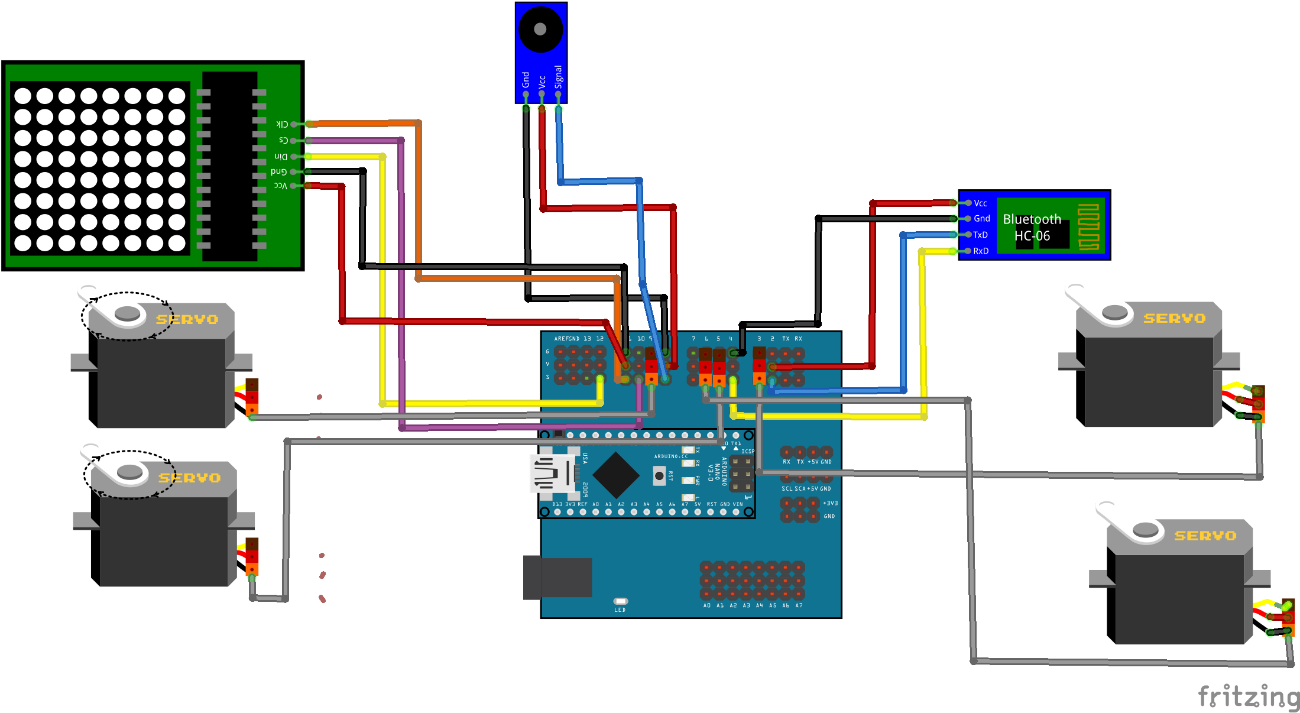
**Download**

<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:**

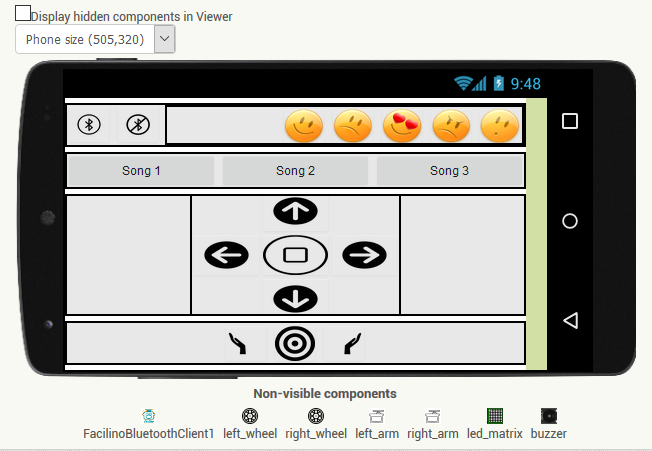
* **Left and right wheel servo motors connected to pins D9 and D5.**
* **Left and right arm servo motors connected to pins D6 and D3.**
* **LED Matrix connected to pins D10 (CS), D11 (CLK), D12 (DIN).**
* **Buzzer module connected to pin D8.**
* **Bluetooth module connected to pin D2 (TX) and pin D3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

****

**App Inventor**

**Designer instructions**

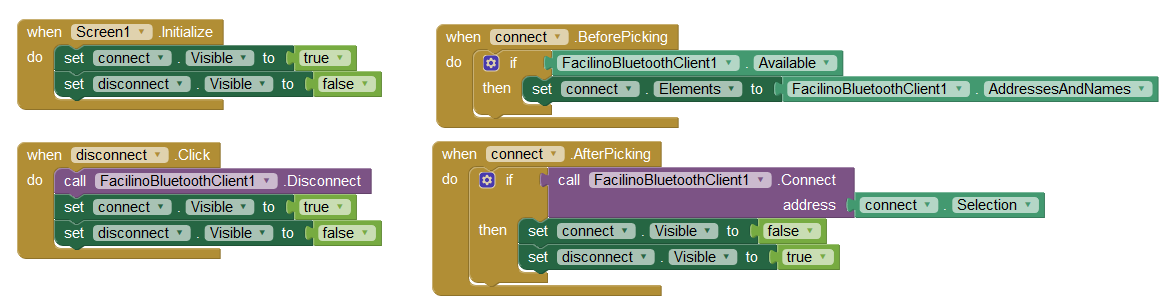
* **Set the ‘*ScreenOrientation’* propery of ‘*Screen1’* component to *Landscape*.**
* **Create a HorizontalArragement component and set the ‘*Width’* property to fill its parent and the ‘*Height’* property to 15%. Inside this component create a ListPicker component and set its ‘*Height’* property to 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 HorizontalArragement component, nested inside the previous HorizontalArragement component. And inside the nested HorizontalArragement 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 HorizontalArragement 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 TableArragement 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 HorizontalArragement 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 tree 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.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 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 the FacilinoBluetoothClient component. Also modify the ‘*Pin’* property to number 6 and 3, respectively.**
* **Create a LEDMatrix8x8Bluetooth components and rename them as ‘led\_matrix’ and assign the property ‘*FacilinoBluetoothClient’* to point 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 the FacilinoBluetoothClient component. Also modify the ‘*Pin’* property to number 8.**

****

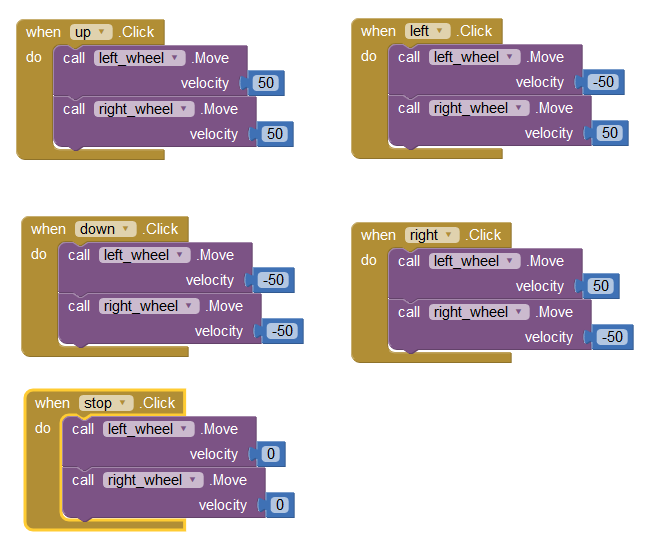
**Blocks**

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

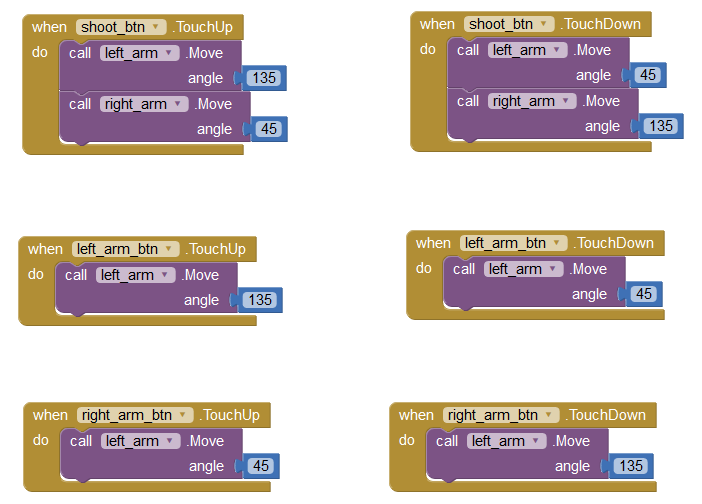
**Connection and disconnection use the ‘*Connect’* and ‘*Disconnect’* methods of the FacilinoBluetoothClient component. To list the MAC address in the ListPicker component, use the ‘*AddressesAndNames’* property. To check if Bluetooth is ‘*Enabled’* and ‘*Available’*, use the corresponding properties.**

****

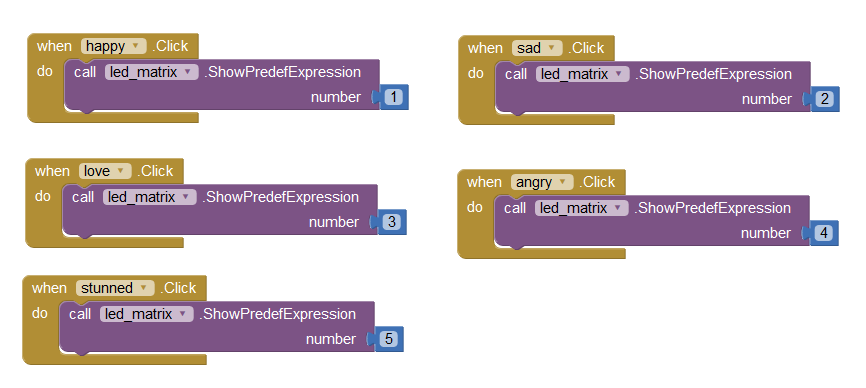
**To set the movements of the robot, we call the ‘Move’ method of ServoContBluetooth components inside the ‘*Click’* event on each Button component. The ‘*velocity’* input argument of the ‘*Move’* method is expressed in a percentage can vary between -100 and 100. So, for instance, in order to move forward,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

****

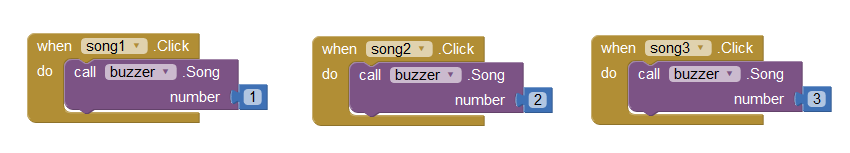
**Similarly, the servos for robot arms/gripper can be moved. To do so, we call the ‘*Move’* method of the ServoBluetooth components and provide the ‘*angle’* in degrees and varies between 0 and 180.**

****

**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**

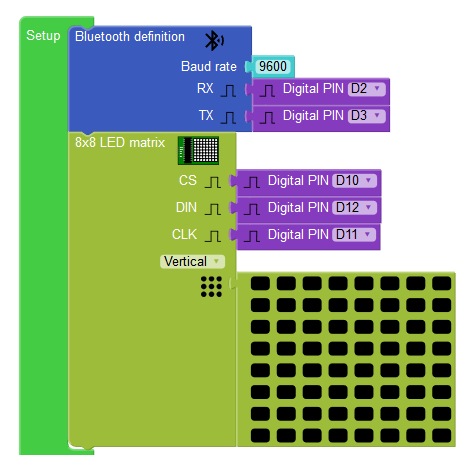
**The *Bluetooth definition* instruction uses pins D2 and D3, to stablish the software serial connection (please, consider that RX and TX between the Bluetooth module and Arduino are crossed). Also, in the setup of the program, use the 8x8 LED matrix instruction with a blank expression to initialise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

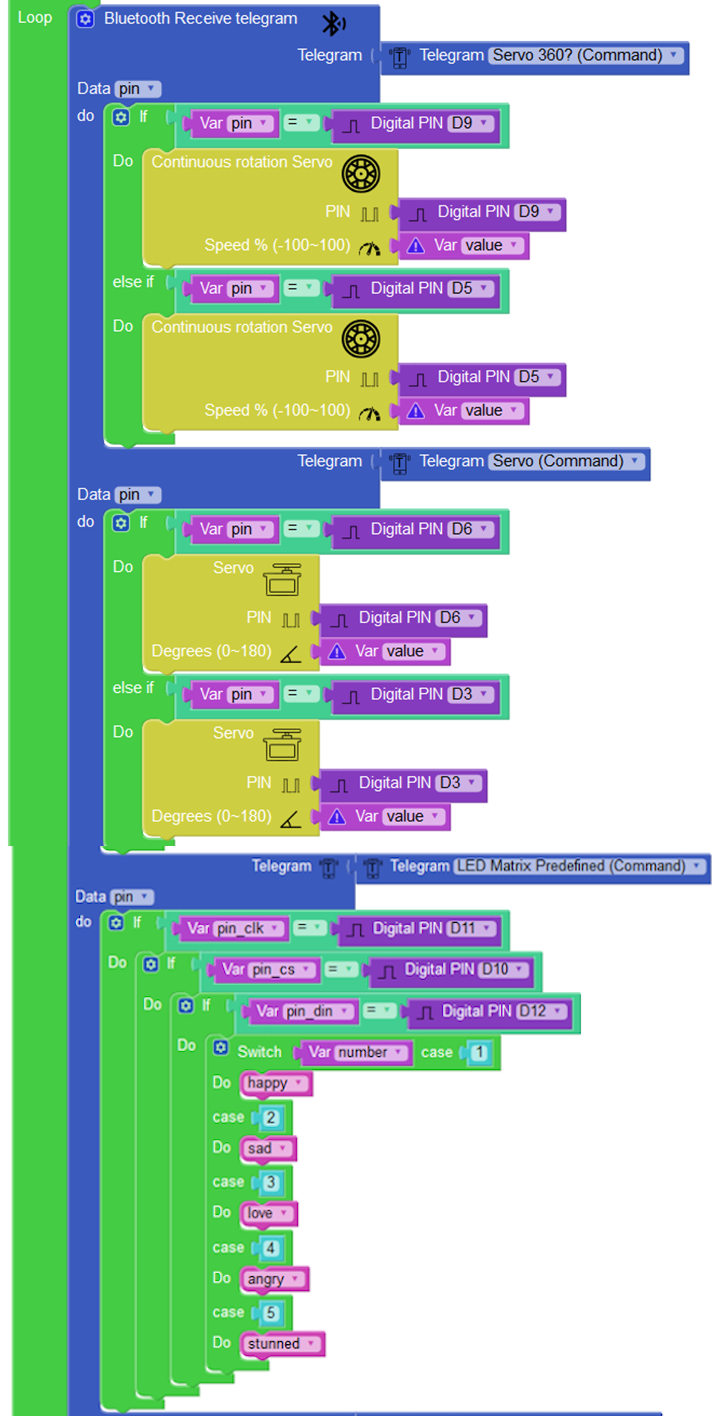
**The *Bluetooth Receive telegram* instruction decodes any received telegram. In this case, we are decoding *Servo 360*, *Servo*, *LED Matrix Predefined* and *Buzzer Melody* telegrams. We must check the pin numbers before using the corresponding instructions on each telegram.**

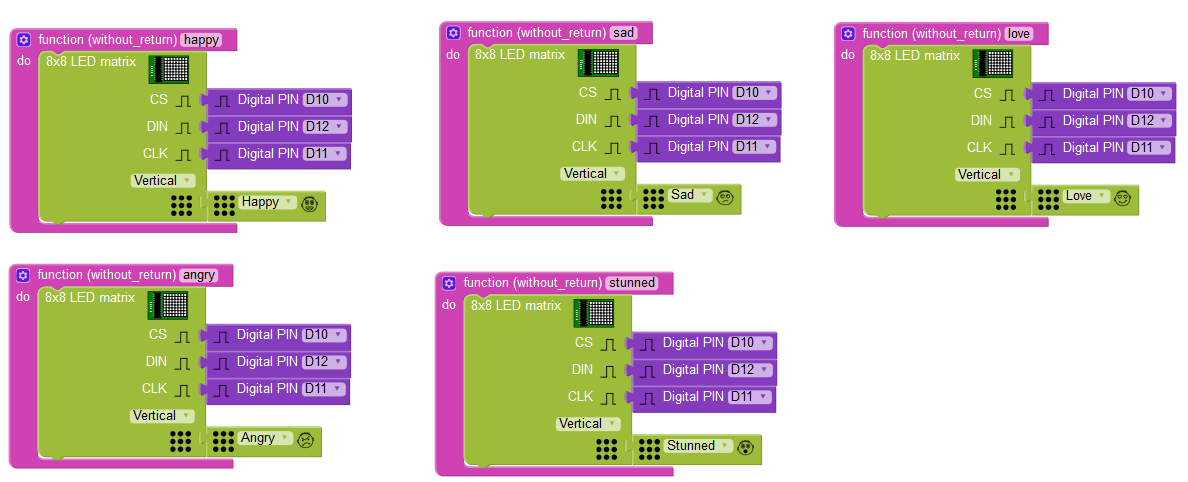
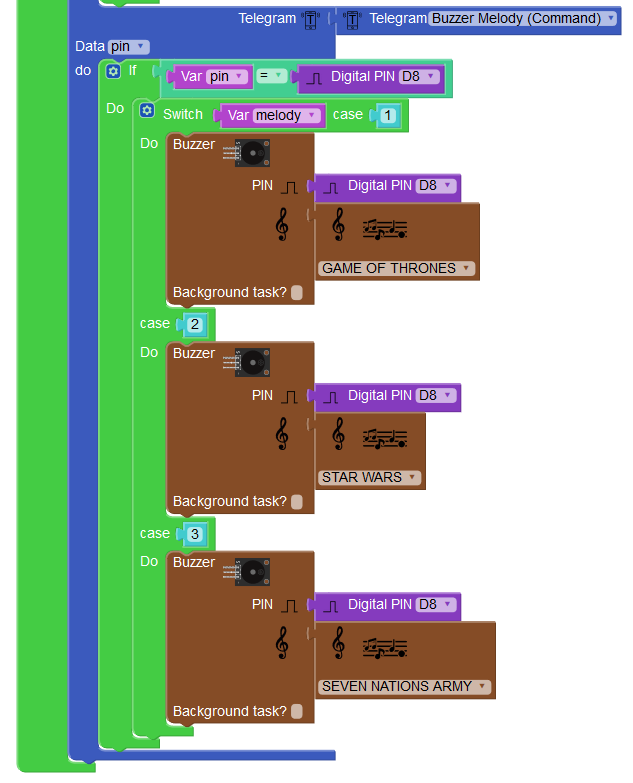
**In particular, we start with the *Servo 360* telegram, that decodes the corresponding telegram and if the pin numbers are correct, then we use the *Continuous rotation Servo* instruction to set the velocity based on the received value. Also, we can use a similar procedure to set the position of robot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

****

****

****

**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 form 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.**

**#include <SoftwareSerial.h>**

**#include <Servo.h>**

**#define BT\_TX\_PIN 2 //TX pin of bluetooth module**

**#define BT\_RX\_PIN 3 //RX pin of bluetooth module**

**#define LEFT\_WHEEL\_PIN 9**

**#define RIGHT\_WHEEL\_PIN 5**

**#define LEFT\_ARM\_PIN 6**

**#define RIGHT\_ARM\_PIN 3**

**#define CLK\_PIN 11**

**#define CS\_PIN 10**

**#define DIN\_PIN 12**

**#define BUZZER\_PIN 8**

**SoftwareSerial \_bt\_device(BT\_TX\_PIN,BT\_RX\_PIN);**

**Servo leftWheel;**

**Servo rightWheel;**

**Servo leftArm;**

**Servo rightArm;**

**int \_bt\_pos=0;**

**unsigned char \_bt\_cmd=0;**

**int \_bt\_length=0;**

**unsigned char \_bt\_data[255];**

**#define CMD\_SERVO 0x10**

**#define CMD\_SERVO\_CONT 0x11**

**#define CMD\_BUZZER\_MELODY 0x21**

**#define CMD\_LED\_MATRIX\_PREDEF\_EXPR 0x51**

**const uint16\_t GAME\_THRONES[] = {1568,562,1047,562,1245,93,1397,93,1568,375,1047,375,1245,93,1397,93,1175,1875,1397,562,932,562,1245,93,1175,93,1397,375,932,562,1245,93,1175,93,1047,750};**

**const uint16\_t IMPERIAL\_MARCH[] = {440,375,440,375,440,375,349,281,523,93,440,375,349,281,523,93,440,750,659,375,659,375,659,375,698,281,523,93,415,375,349,281,523,93,440,750,880,375,440,281,440,93,880,375,831,281,784,93,740,93,698,93,740,187,0,187,446,281,622,375,587,281,554,93,523,93,494,93,523,187,0,187,349,187,415,375,349,281,392,93,523,375,440,281,523,93,659,750,880,375,440,281,440,93,880,375,831,281,784,93,740,93,698,93,740,187,0,187,446,281,622,375,587,281,554,93,523,93,494,93,523,187,0,187,330,187,415,375,330,281,523,93,440,375,349,281,523,93,440,750};**

**const uint16\_t SEVEN\_NATIONS\_ARMY[] = {330,759,330,252,392,378,330,125,0,252,294,252,262,1012,245,1012,330,759,330,252,392,378,330,125,0,252,294,252,262,506,294,252,262,252,245,1012,330,759,330,252,392,378,330,125,0,252,294,252,262,1012,245,1012};**

**void LEDMatrix\_init(int cs, int din, int clk);**

**void writeRow(int cs, int din, int clk, int row, int data);**

**void maxAll (int cs, int din, int clk, int reg, int col);**

**void putByte (int din, int clk, int data);**

**void expression(int cs, int din, int clk, int col1, int col2, int col3, int col4, int col5, int col6, int col7, int col8);**

**void playMelody(int pin,const uint16\_t\* melody, int length);**

**void happy ();**

**void sad ();**

**void love ();**

**void angry ();**

**void stunned ();**

**void setup()**

**{**

**\_bt\_device.begin(9600);**

**\_bt\_device.flush();**

**pinMode(BUZZER\_PIN,OUTPUT);**

**pinMode(CLK\_PIN,OUTPUT);**

**pinMode(CS\_PIN,OUTPUT);**

**pinMode(DIN\_PIN,OUTPUT);**

**//Set the LED Matrix component**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,11,7);**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,9,0);**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,12,1);**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,15,0);**

**for (int i = 1; i <= 8; i++)**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,i,0);**

**maxAll(CS\_PIN,DIN\_PIN,CLK\_PIN,10,15);**

**expression(CS\_PIN,DIN\_PIN,CLK\_PIN,0,0,0,0,0,0,0,0);**

**leftWheel.attach(LEFT\_WHEEL\_PIN);**

**delay(300);**

**rightWheel.attach(RIGHT\_WHEEL\_PIN);**

**delay(300);**

**leftArm.attach(LEFT\_ARM\_PIN);**

**delay(300);**

**rightArm.attach(RIGHT\_ARM\_PIN);**

**delay(300);**

**}**

**void loop()**

**{**

**if (\_bt\_device.available()>0)**

**{**

**unsigned char c;**

**\_bt\_device.readBytes(&c,1);**

**if ((c=='@')&&(\_bt\_pos==0))**

**\_bt\_pos++;**

**else if (\_bt\_pos==1){**

**\_bt\_pos++;**

**\_bt\_cmd=c;**

**}**

**else if (\_bt\_pos==2){**

**\_bt\_pos++;**

**\_bt\_length=c;**

**}**

**else if ((\_bt\_pos>=3)&&(\_bt\_pos<=(\_bt\_length+2))){**

**\_bt\_data[\_bt\_pos-3]=c;**

**\_bt\_pos++;**

**}**

**else if ((\_bt\_pos==(\_bt\_length+3))&&(c=='\*')){**

**if (\_bt\_cmd==CMD\_SERVO\_CONT){**

**int pin = \_bt\_data[0];**

**byte value = \_bt\_data[1];**

**if (pin==LEFT\_WHEEL\_PIN)**

**leftWheel.write** **(((value\*90)/100+90));**

**else if (pin==RIGHT\_WHEEL\_PIN)**

**rightWheel.write** **(((value\*90)/100+90));**

**}**

**if (\_bt\_cmd==CMD\_SERVO){**

**int pin = \_bt\_data[0];**

**byte value = \_bt\_data[1];**

**if (pin==LEFT\_ARM\_PIN)**

**leftArm.write(value);**

**else if (pin==RIGHT\_ARM\_PIN)**

**rightArm.write(value);**

**}**

**if (\_bt\_cmd==CMD\_LED\_MATRIX\_PREDEF\_EXPR){**

**int pin\_clk = \_bt\_data[0];**

**int pin\_cs = \_bt\_data[1];**

**int pin\_din = \_bt\_data[2];**

**byte number = \_bt\_data[3];**

**if ((pin\_clk==CLK\_PIN)&&(pin\_cs==CS\_PIN)&&(pin\_din==DIN\_PIN)){**

**switch (number)**

**{**

**case 1:**

**happy();**

**break;**

**case 2:**

**sad();**

**break;**

**case 3:**

**love();**

**break;**

**case 4:**

**angry();**

**break;**

**case 5:**

**stunned();**

**break;**

**}**

**}**

**}**

**if (\_bt\_cmd==CMD\_BUZZER\_MELODY){**

**int pin = \_bt\_data[0];**

**byte melody = \_bt\_data[1];**

**if (pin==BUZZER\_PIN){**

**switch (melody)**

**{**

**case 1:**

**playMelody(BUZZER\_PIN,GAME\_THRONES,sizeof(GAME\_THRONES)/(2\*sizeof(uint16\_t)));**

**break;**

**case 2:**

**playMelody(BUZZER\_PIN,IMPERIAL\_MARCH,sizeof(IMPERIAL\_MARCH)/(2\*sizeof(uint16\_t)));**

**break;**

**case 3:**

**playMelody(BUZZER\_PIN,SEVEN\_NATIONS\_ARMY,sizeof(SEVEN\_NATIONS\_ARMY)/(2\*sizeof(uint16\_t)));**

**break;**

**}**

**}**

**}**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**else{**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**}**

**}**

**void LEDMatrix\_init(int cs, int din, int clk) {**

**maxAll(cs,din,clk,11,7);**

**maxAll(cs,din,clk,9,0);**

**maxAll(cs,din,clk,12,1);**

**maxAll(cs,din,clk,15,0);**

**for (int i = 1; i <= 8; i++)**

**maxAll(cs,din,clk,i, 0);**

**maxAll(cs,din,clk,10,15);**

**}**

**void writeRow(int cs, int din, int clk, int row, int data) {**

**digitalWrite(cs,LOW);**

**putByte(din,clk,row);**

**putByte(din,clk,data);**

**digitalWrite(cs,LOW);**

**digitalWrite(cs,HIGH);**

**}**

**void maxAll (int cs, int din, int clk, int reg, int col) {**

**digitalWrite(cs,LOW);**

**putByte(din,clk,reg);**

**putByte(din,clk,col);**

**digitalWrite(cs,LOW);**

**digitalWrite(cs,HIGH);**

**}**

**void putByte (int din, int clk, int data) {**

**byte i = 8;**

**byte mask;**

**while(i > 0) {mask = 0x01 << (i - 1);**

**digitalWrite(clk,LOW);**

**if (data & mask)**

**{**

**digitalWrite(din,HIGH);**

**}else{**

**digitalWrite(din,LOW);**

**}**

**digitalWrite(clk,HIGH);**

**--i;**

**}**

**}**

**void expression(int cs, int din, int clk, int col1, int col2, int col3, int col4, int col5, int col6, int col7, int col8) {**

**writeRow(cs,din,clk,1, col1);**

**writeRow(cs,din,clk,2, col2);**

**writeRow(cs,din,clk,3, col3);**

**writeRow(cs,din,clk,4, col4);**

**writeRow(cs,din,clk,5, col5);**

**writeRow(cs,din,clk,6, col6);**

**writeRow(cs,din,clk,7, col7);**

**writeRow(cs,din,clk,8, col8);**

**}**

**void playMelody(int pin,const uint16\_t\* melody, int length)**

**{**

**unsigned int note;**

**unsigned long duration;**

**uint16\_t\* melody\_ptr=(uint16\_t\*)melody;**

**for (int i=0;i<length;i++)**

**{**

**note=\*melody\_ptr++;**

**duration=\*melody\_ptr++;**

**tone(pin,note,duration);**

**delay(duration);**

**noTone(pin);**

**}**

**}**

**void happy () {**

**expression(10,12,11,12,24,48,48,48,48,24,12);**

**}**

**void sad () {**

**expression(10,12,11,48,24,12,12,12,12,24,48);**

**}**

**void love () {**

**expression(10,12,11,12,30,62,124,124,62,30,12);**

**}**

**void angry () {**

**expression(10,12,11,2,68,36,16,16,36,68,2);**

**}**

**void stunned () {**

**expression(10,12,11,0,0,0,24,24,0,0,0);**

**}**

## **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.

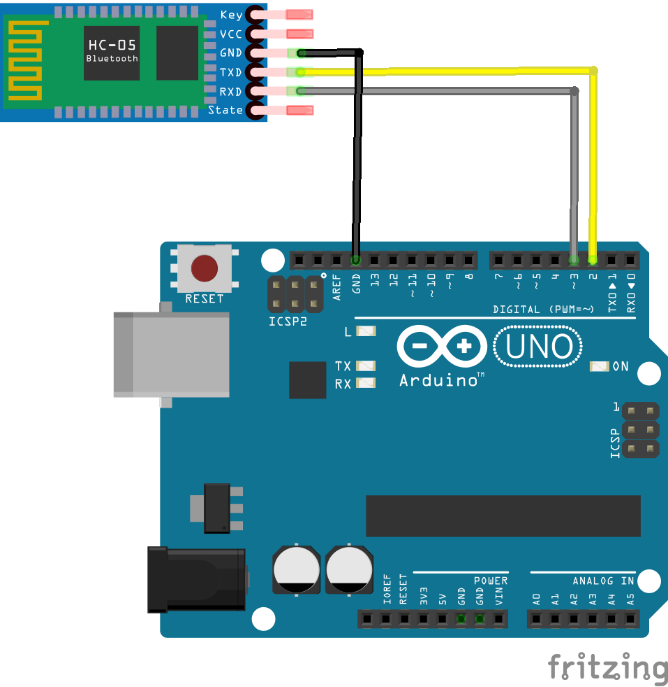
**Download**

<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**

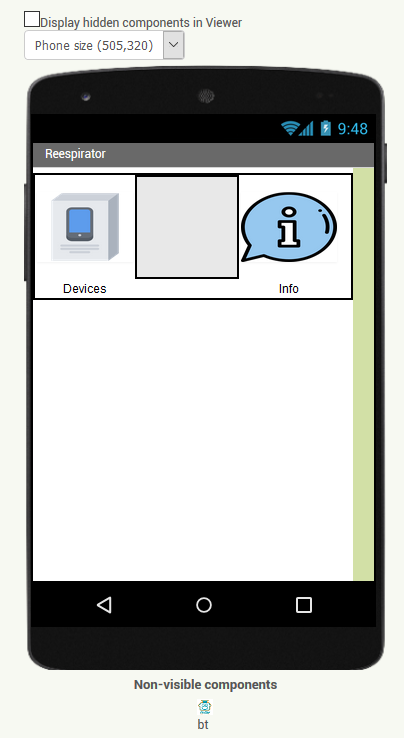
**For this App, we have several screens. One of the common problems when dealing with multiple screens with the standard BluetoothClient component included in App Inventor 2 is that it gets disconnected. This is easily solved with the FacilinoBluetoothClient component, due to it’s ‘*Reconnect’* method. 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**

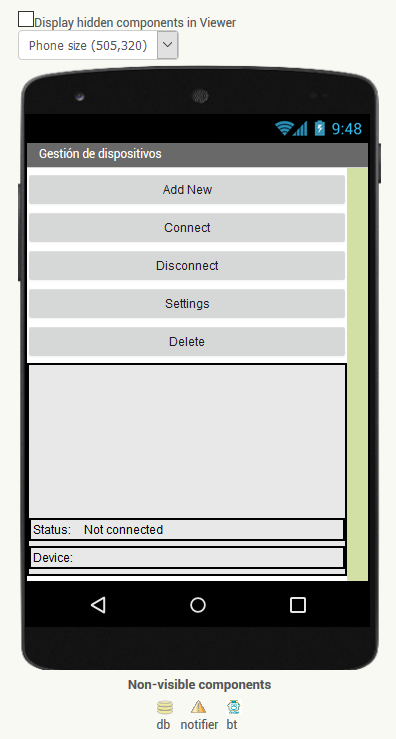
**When each of the Button component is click (‘*Click’* event), we call the procedure to disconnect the bluetooth device and then we open the other screen. The FacilinoBluetoothClient is included here to keep connection, although it is not strictly necessary. In addition to this, when that screen is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

****

**Devices Designer**

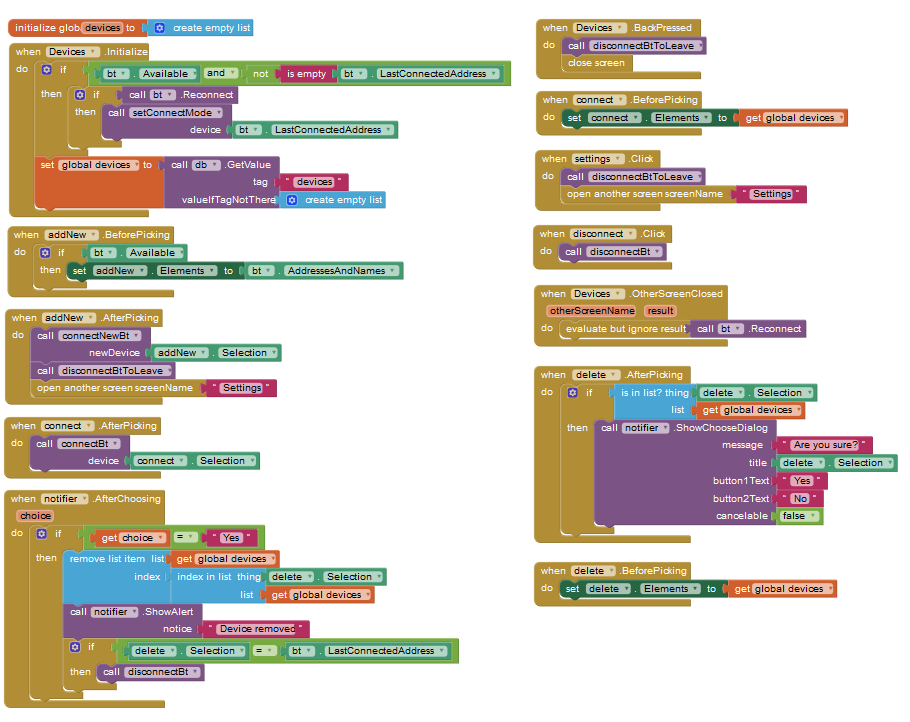
**A UI with several Button and ListPicker components to manage all connection issues. It also includes a FacilinoBluetoothClient, a TinyDB and Notifier components.**

|  |  |  |
| --- | --- | --- |
| **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.** |

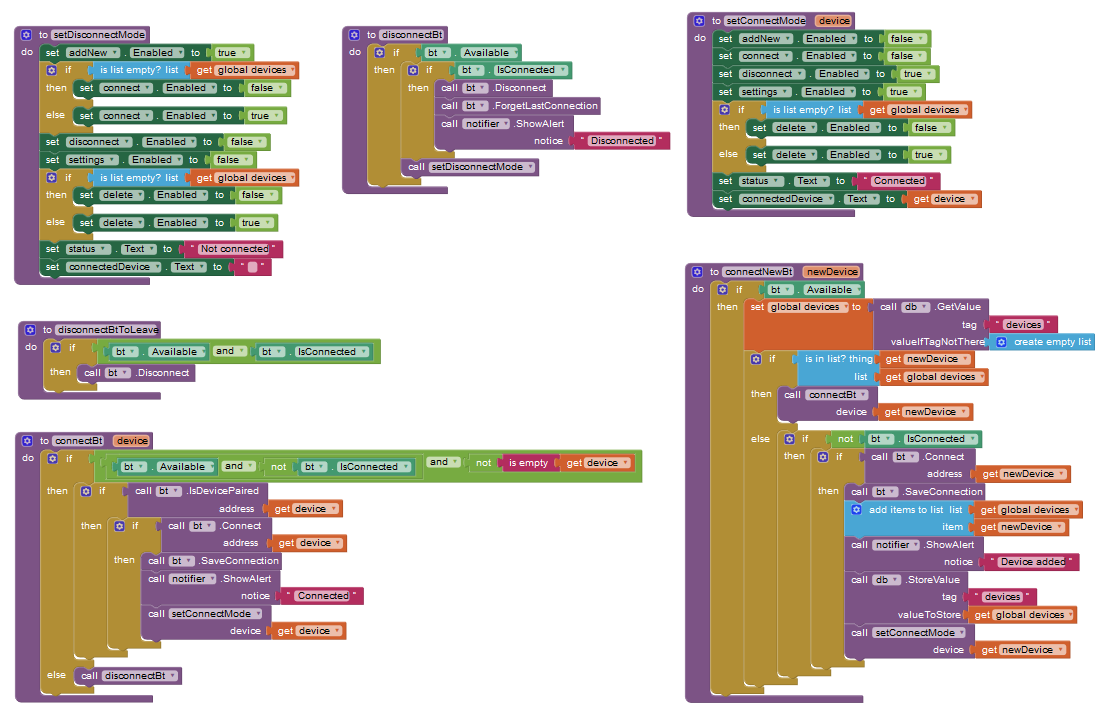
****

**Devices Blocks**

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

****

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

****

**Settings Designer**

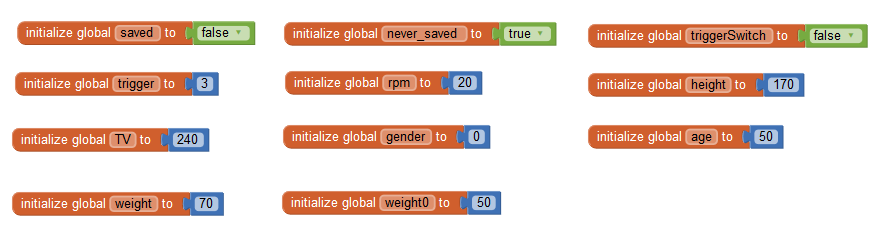
**This UI contains a combination of Button, TextBox, Slider, Label, Spinner, Switch components to display patient data as well as to read, save and restore data. It also includes a FacilinoBluetootClient with a set of IntVariableBluetooth, BooleanVariableBluetooth and StringVariableBluetooth components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

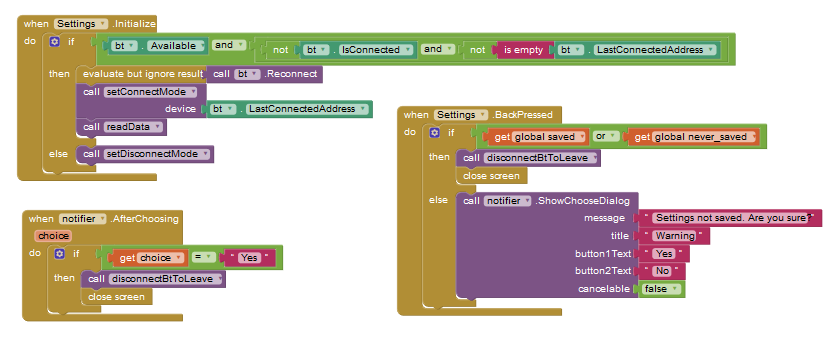
****

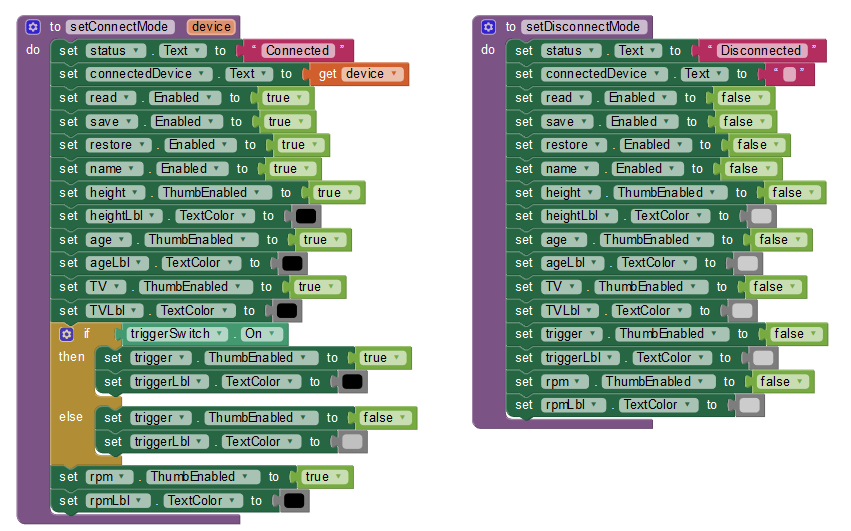
**Settings Blocks**

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

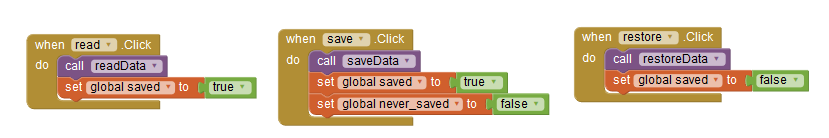
****

**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.**

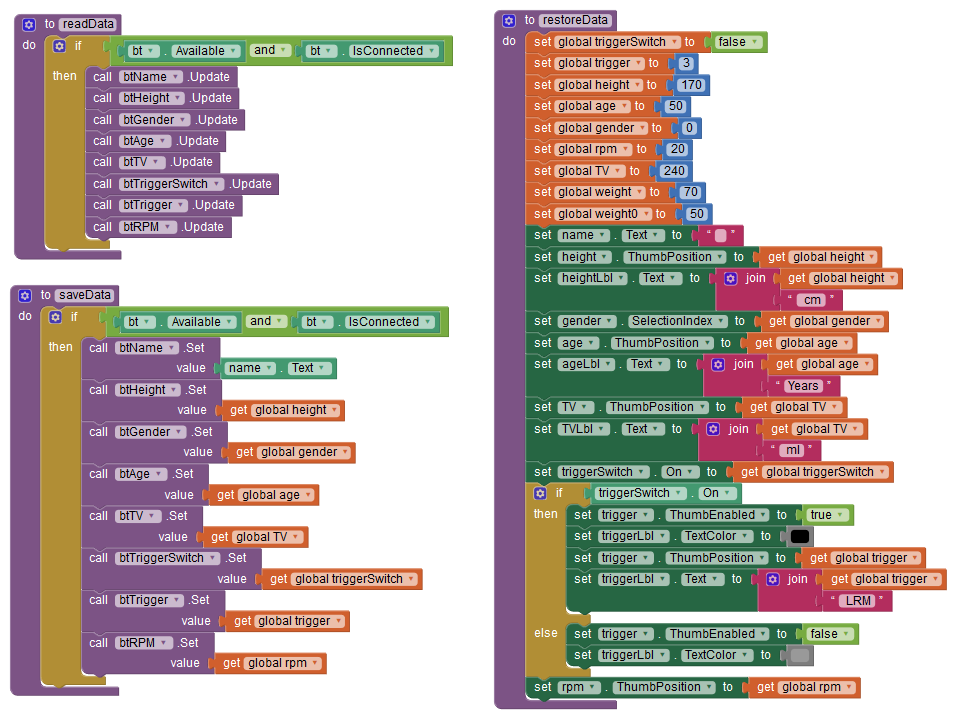
****

****

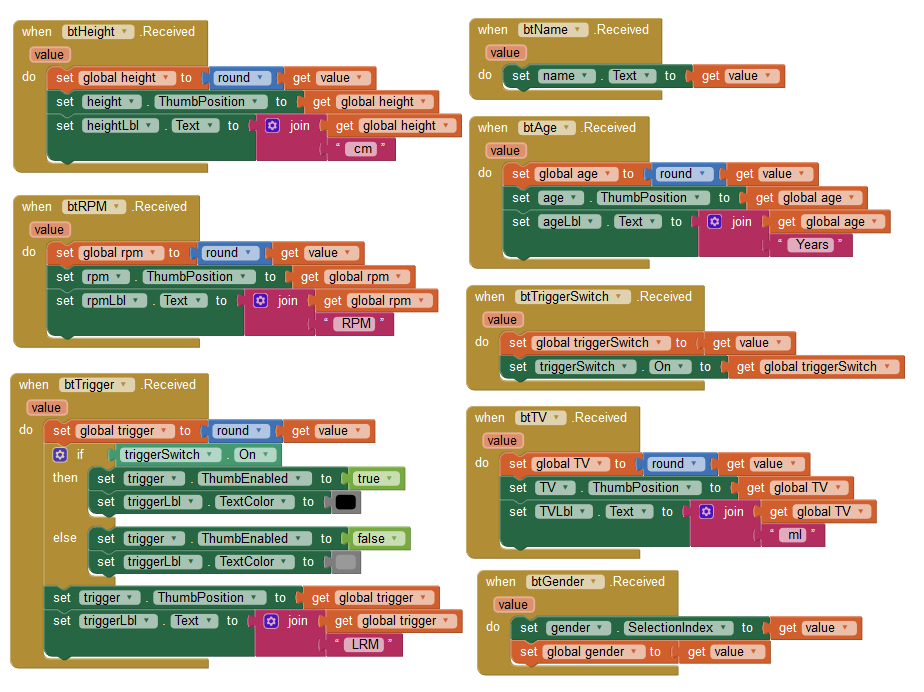
**These three ‘*Click’* events call procedures that actually retrieve, save or restore default values of data:**

****

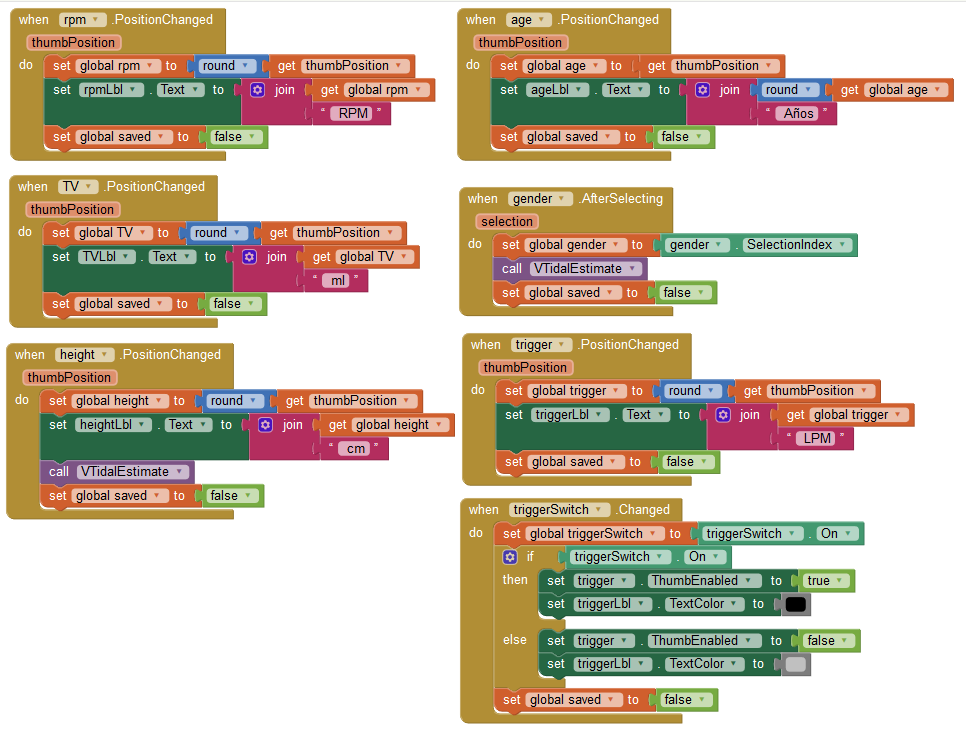
**These are the procedures used in the previous ‘*Click’* events.**

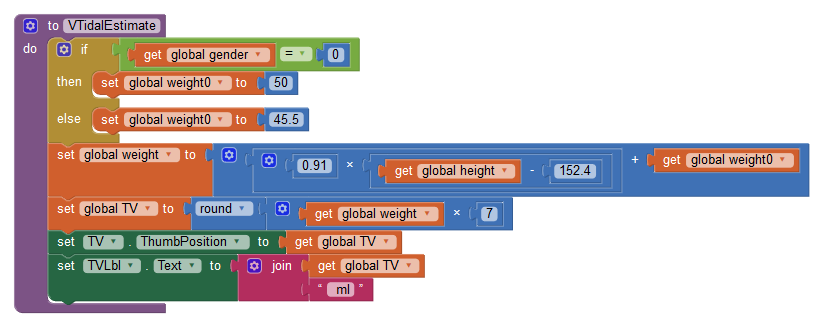
****

**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.**

****

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

****

****

**Info Designer**

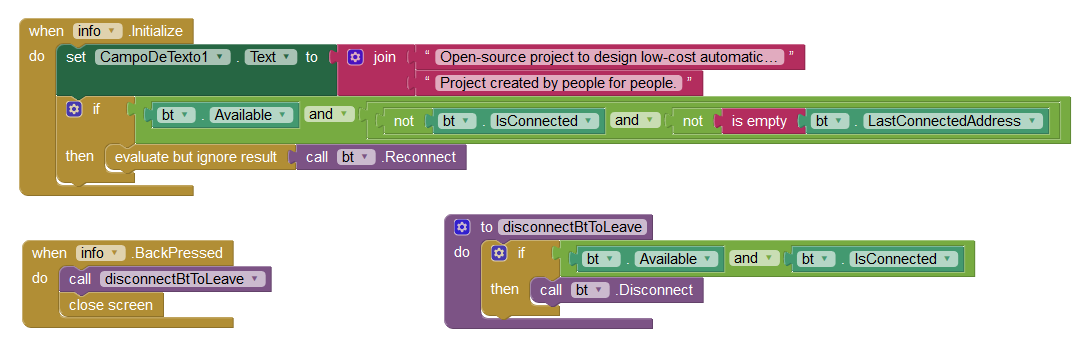
**A very simple UI to show info about this project. The FacilinoBluetoothClient is included here to keep connection, although it is not strictly necessary.**

|  |  |  |
| --- | --- | --- |
| **Info** | | |
| **Name** | **Component Type** | **Description** |
| **bt** | **FacilinoBluetoothClient** | **Bluetooth Client.** |

****

**Info Blocks**

**This code handles reconnection and disconnection.**

****

**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 form 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.**

**#include <SoftwareSerial.h>**

**#define BT\_TX\_PIN 2 //TX pin of bluetooth module**

**#define BT\_RX\_PIN 3 //RX pin of bluetooth module**

**//Maximum number of variables to store for receive/transmitting configuration**

**#define MAX\_BOOLs 1**

**#define MAX\_INTs 20**

**#define MAX\_FLOATs 0**

**#define MAX\_STRINGs 1**

**#define PROTOCOL\_DEBUG //Uncomment for debug**

**#define CMD\_BOOLEAN\_VAR\_WRITE\_REQ 0x80**

**#define CMD\_BOOLEAN\_VAR\_READ\_REQ 0x81**

**#define CMD\_BOOLEAN\_VAR\_READ\_RESP 0x82**

**#define CMD\_INT\_VAR\_WRITE\_REQ 0x83**

**#define CMD\_INT\_VAR\_READ\_REQ 0x84**

**#define CMD\_INT\_VAR\_READ\_RESP 0x85**

**#define CMD\_FLOAT\_VAR\_WRITE\_REQ 0x86**

**#define CMD\_FLOAT\_VAR\_READ\_REQ 0x87**

**#define CMD\_FLOAT\_VAR\_READ\_RESP 0x88**

**#define CMD\_STRING\_VAR\_WRITE\_REQ 0x89**

**#define CMD\_STRING\_VAR\_READ\_REQ 0x8A**

**#define CMD\_STRING\_VAR\_READ\_RESP 0x8B**

**bool boolVars[MAX\_BOOLs];**

**int intVars[MAX\_INTs];**

**float floatVars[MAX\_FLOATs];**

**String stringVars[MAX\_STRINGs];**

**//Position of boolean variables**

**#define VAR\_TRIGGER\_SWITCH 0**

**//Position of integer variables**

**#define VAR\_HEIGHT 0**

**#define VAR\_GENDER 1**

**#define VAR\_AGE 2**

**#define VAR\_VTIDAL 3**

**#define VAR\_TRIGGER 4**

**#define VAR\_RPM 5**

**//Position of string variables**

**#define VAR\_NAME 0**

**//Variables related with the bluetooth protocol**

**SoftwareSerial \_bt\_device(BT\_TX\_PIN,BT\_RX\_PIN);**

**int \_bt\_pos=0;**

**unsigned char \_bt\_cmd=0;**

**int \_bt\_length=0;**

**unsigned char \_bt\_data[255];**

**void parseBtTelegram();**

**void setup()**

**{**

**//Initialise bluetooth device**

**Serial.begin(115200);**

**\_bt\_device.begin(9600);**

**\_bt\_device.flush();**

**intVars[VAR\_HEIGHT]=165;**

**intVars[VAR\_GENDER]=1;**

**intVars[VAR\_AGE]=15;**

**intVars[VAR\_VTIDAL]=240;**

**intVars[VAR\_TRIGGER]=5;**

**intVars[VAR\_RPM]=19;**

**stringVars[VAR\_NAME]="Pepe";**

**}**

**void loop()**

**{**

**//IMPORTANT: To properly work, we must ensure that the rest of the code**

**// allows this routine to execute regularly, otherwise, it won't be**

**// responsive**

**while (\_bt\_device.available()>0)**

**parseBtTelegram();**

**}**

**void parseBtTelegram()**

**{**

**unsigned char c;**

**\_bt\_device.readBytes(&c,1);**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print((byte)c);**

**Serial.print(" ");**

**#endif**

**if ((c=='@')&&(\_bt\_pos==0))**

**\_bt\_pos++;**

**else if (\_bt\_pos==1) {**

**\_bt\_pos++;**

**\_bt\_cmd=c;**

**} else if (\_bt\_pos==2) {**

**\_bt\_pos++;**

**\_bt\_length=c;**

**} else if ((\_bt\_pos>=3)&&(\_bt\_pos<=(\_bt\_length+2))) {**

**\_bt\_data[\_bt\_pos-3]=c;**

**\_bt\_pos++;**

**} else if ((\_bt\_pos==(\_bt\_length+3))&&(c=='\*')){**

**#ifdef PROTOCOL\_DEBUG**

**Serial.println(".");**

**#endif**

**if (\_bt\_cmd==CMD\_BOOLEAN\_VAR\_READ\_REQ){ //Boolean read request**

**int index = \_bt\_data[0];**

**int value = (int)boolVars[index];**

**#ifdef PROTOCOL\_DEBUG**

**if (index==VAR\_TRIGGER\_SWITCH) {**

**Serial.print(value?1:0);**

**Serial.print(" ");**

**Serial.println("Trigger switch read requested");**

**}**

**#endif**

**//Boolean read response**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_BOOLEAN\_VAR\_READ\_RESP);**

**\_bt\_device.write((byte)2);**

**\_bt\_device.write((byte)index);**

**\_bt\_device.write((byte)(value?1:0));**

**\_bt\_device.write('\*');**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print((byte)'@'); Serial.print(" ");**

**Serial.print((byte)CMD\_BOOLEAN\_VAR\_READ\_RESP); Serial.print(" ");**

**Serial.print((byte)2); Serial.print(" ");**

**Serial.print((byte)index); Serial.print(" ");**

**Serial.print((byte)((boolVars[index])?1:0)); Serial.print(" ");**

**Serial.println((byte)'\*');**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_BOOLEAN\_VAR\_WRITE\_REQ){ //Boolean write request**

**int index = \_bt\_data[0];**

**bool value=(bool)(\_bt\_data[1]==1?true:false);**

**boolVars[index]=value;**

**#ifdef PROTOCOL\_DEBUG**

**if (index==VAR\_TRIGGER\_SWITCH){**

**Serial.print(value?1:0);**

**Serial.print(" ");**

**Serial.println("Trigger switch write requested");**

**}**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_INT\_VAR\_READ\_REQ){ //Int read request**

**int index = \_bt\_data[0];**

**int value = intVars[index];**

**#ifdef PROTOCOL\_DEBUG**

**if (index==VAR\_HEIGHT){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Height read requested");**

**}**

**else if (index==VAR\_GENDER){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Gender read requested");**

**} else if (index==VAR\_AGE) {**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Age read requested");**

**} else if (index==VAR\_VTIDAL){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("VTidal read requested");**

**} else if (index==VAR\_TRIGGER){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Trigger read requested");**

**} else if (index==VAR\_RPM){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("RPM read requested");**

**}**

**#endif**

**//Int read response**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_INT\_VAR\_READ\_RESP);**

**\_bt\_device.write((byte)3);**

**\_bt\_device.write((byte)index);**

**\_bt\_device.write((byte)((value&0xFF00)>>8));**

**\_bt\_device.write((byte)(value&0x00FF));**

**\_bt\_device.write('\*');**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print((byte)'@'); Serial.print(" ");**

**Serial.print((byte)CMD\_INT\_VAR\_READ\_RESP); Serial.print(" ");**

**Serial.print((byte)3); Serial.print(" ");**

**Serial.print((byte)index); Serial.print(" ");**

**Serial.print((byte)((value&0xFF00)>>8)); Serial.print(" ");**

**Serial.print((byte)(value&0x00FF)); Serial.print(" ");**

**Serial.println((byte)'\*');**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_INT\_VAR\_WRITE\_REQ){ //Int write request**

**int index = \_bt\_data[0];**

**int value = (int)((((int)\_bt\_data[1]<<8)&0xFF00)|(((int)\_bt\_data[2])&0x00FF));**

**intVars[index]=value;**

**#ifdef PROTOCOL\_DEBUG**

**if (index==VAR\_HEIGHT){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Height write requested");**

**} else if (index==VAR\_GENDER){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Gender write requested");**

**} else if (index==VAR\_AGE){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Age write requested");**

**} else if (index==VAR\_VTIDAL){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("VTidal write requested");**

**} else if (index==VAR\_TRIGGER){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("Trigger write requested");**

**} else if (index==VAR\_RPM){**

**Serial.print(value);**

**Serial.print(" ");**

**Serial.println("RPM write requested");**

**}**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_FLOAT\_VAR\_READ\_REQ){ //Float read request**

**int index = \_bt\_data[0];**

**float value = floatVars[index];**

**int\* valuePtr = (int\*)&value;**

**//Float read response**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_FLOAT\_VAR\_READ\_RESP);**

**\_bt\_device.write((byte)5);**

**\_bt\_device.write((byte)index);**

**\_bt\_device.write((byte)((\*valuePtr&0xFF000000)>>24));**

**\_bt\_device.write((byte)((\*valuePtr&0x00FF0000)>>16));**

**\_bt\_device.write((byte)((\*valuePtr&0x0000FF00)>>8));**

**\_bt\_device.write((byte)(\*valuePtr&0x000000FF));**

**\_bt\_device.write('\*');**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print((byte)'@'); Serial.print(" ");**

**Serial.print((byte)CMD\_FLOAT\_VAR\_READ\_RESP); Serial.print(" ");**

**Serial.print((byte)5); Serial.print(" ");**

**Serial.print((byte)index); Serial.print(" ");**

**Serial.print((byte)((\*valuePtr&0xFF000000)>>24)); Serial.print(" ");**

**Serial.print((byte)((\*valuePtr&0x00FF0000)>>16)); Serial.print(" ");**

**Serial.print((byte)((\*valuePtr&0x0000FF00)>>8)); Serial.print(" ");**

**Serial.print((byte)(\*valuePtr&0x000000FF)); Serial.print(" ");**

**Serial.println((byte)'\*');**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_FLOAT\_VAR\_WRITE\_REQ){ //Float write request**

**int index = \_bt\_data[0];**

**int valueH=((int)\_bt\_data[1]<<8)&0xFF00)|((int)\_bt\_data[1]);**

**int valueL=((int)\_bt\_data[2]<<8)&0xFF00)|((int)\_bt\_data[3]);**

**int value==((int)((valueH<<16)&0xFFFF0000))|((int)valueL&0x0000FFFF);**

**float \*valuePtr = (float\*)&value;**

**floatVars[index]=\*valuePtr;**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print(\*valuePtr);**

**Serial.println(" ");**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_STRING\_VAR\_READ\_REQ){ //String read request**

**int index = \_bt\_data[0];**

**char\* str = stringVars[index];**

**String Str(str);**

**#ifdef PROTOCOL\_DEBUG**

**if (index==VAR\_NAME){**

**Serial.print(Str);**

**Serial.print(" ");**

**Serial.println("name read requested");**

**}**

**#endif**

**int len = Str.length();**

**\_bt\_device.write('@');**

**\_bt\_device.write((byte)CMD\_STRING\_VAR\_READ\_RESP);**

**\_bt\_device.write((byte)2+len);**

**\_bt\_device.write((byte)index);**

**\_bt\_device.write((byte)len);**

**for (int i=0;i<len;i++)**

**\_bt\_device.write(Str.charAt(i));**

**\_bt\_device.write('\*');**

**#ifdef PROTOCOL\_DEBUG**

**Serial.print((byte)'@'); Serial.print(" ");**

**Serial.print((byte)CMD\_STRING\_VAR\_READ\_RESP); Serial.print(" ");**

**Serial.print((byte)2+len); Serial.print(" ");**

**Serial.print((byte)index); Serial.print(" ");**

**Serial.print((byte)len); Serial.print(" ");**

**for (int i=0;i<len;i++){**

**Serial.print(str.charAt(i)); Serial.print(" ");}**

**Serial.println((byte)'\*');**

**#endif**

**}**

**else if (\_bt\_cmd==CMD\_STRING\_VAR\_WRITE\_REQ){ //String write request**

**int index = \_bt\_data[0];**

**char \*str = stringVars[index];**

**for (int i=0;i<\_bt\_length;i++)**

**str[i]=(char)\_bt\_data[i+1];**

**}**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**} else{**

**\_bt\_pos=0;**

**\_bt\_length=0;**

**}**

**}**

**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



**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** | **…** | **DATAN-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 (AppYArduino)**

**Request a digital input read for the indicated digital pin.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **END** |
| **@** | **0x00** | **1** | **Pin** | **\*** |

**Digital Read Response (ArduinoYApp)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **END** |
| **@** | **0x01** | **2** | **Pin** | **Value** | **\*** |

**Digital Write (AppYArduino)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **END** |
| **@** | **0x02** | **2** | **Pin** | **Value** | **\*** |

**Analog Read Request (AppYArduino)**

**Request an analog input read for the indicated analog pin.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **END** |
| **@** | **0x03** | **1** | **Pin** | **\*** |

**Analog Read Response (ArduinoYApp)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1-2** | **END** |
| **@** | **0x04** | **3** | **Pin** | **Value** | **\*** |

**Analog Write (AppYArduino)**

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **DATA2** | **END** |
| **@** | **0x05** | **3** | **Pin** | **Value** | | **\*** |

**Servo (AppYArduino)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **END** |
| **@** | **0x10** | **2** | **Pin** | **Angle** | **\*** |

**Continuous Servo (AppYArduino)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **END** |
| **@** | **0x11** | **2** | **Pin** | **Velocity** | **\*** |

**Sonar Request (AppYArduino)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **END** |
| **@** | **0x12** | **2** | **Echo pin** | **Trigger Pin** | **\*** |

**Sonar Response (ArduinoYApp)**

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1** | **DATA2-3** | **END** |
| **@** | **0x13** | **4** | **Echo pin** | **Trigger Pin** | **Distance** | **\*** |

**Buzzer Tone (AppYArduino)**

**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** | **DATA0** | **DATA1-2** | **DATA3-4** | **END** |
| **@** | **0x20** | **2** | **Pin** | **Frequency** | **Duration** | **\*** |

**Buzzer Melody (AppYArduino)**

**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** | **DATA0** | **DATA1-2** | **DATA3-4** | **…** | **DATAN-4-N-2** | **DATAN-2-N-1** | **END** |
| **@** | **0x21** | **4\*M+1** | **Pin** | **Freq.0** | **Dur.0** |  | **Freq.M-1** | **Dur.M-1** | **\*** |

**DHT Request (AppYArduino)**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **END** |
| **@** | **0x22** | **2** | **Pin** | **\*** |

**DHT Response (ArduinoYApp)**

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1-2** | **DATA3-4** | **END** |
| **@** | **0x23** | **2** | **Pin** | **Temperature** | **Humidity** | **\*** |

**8x8 LED Matrix (AppYArduino)**

**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** | **DATA0** | **DATA1** | **DATA2** | **DATA3** | **…** | **DATA10** | **END** |
| **@** | **0x50** | **11** | **CLK Pin** | **DIN Pin** | **CS Pin** | **Col1** |  | **Col8** | **\*** |

**8x8 LED Matrix Predefined Expression (AppYArduino)**

**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** | **DATA0** | **DATA1** | **DATA2** | **DATA3** | **END** |
| **@** | **0x51** | **4** | **CLK Pin** | **DIN Pin** | **CS Pin** | **Expression** | **\*** |

**RGB LED Strip (AppYArduino)**

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **STX** | **CMD** | **LENGTH** | **DATA0** | **DATA1-3** | **…** | **DATA10** | **END** |
| **@** | **0x60** | **3\*M+1** | **Pin** | **RGB0** |  | **RGBM-1** | **\*** |

**RGB LED Strip Predefined Expression (AppYArduino)**

**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** | **DATA0** | **DATA3** | **END** |
| **@** | **0x61** | **2** | **Pin** | **Expression** | **\*** |

**Sets Boolean Variable (AppYArduino)**

**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** | **DATA0** | **DATA1** | **END** |
| **@** | **0x80** | **2** | **Index** | **Value** | **\*** |

**Request Boolean Variable (AppYArduino)**

**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** | **DATA0** | **END** |
| **@** | **0x81** | **1** | **Index** | **\*** |

**Response Boolean Variable Request (ArduinoYApp)**

**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** | **DATA0** | **DATA1** | **END** |
| **@** | **0x82** | **2** | **Index** | **Value** | **\*** |

**Sets Integer Variable (AppYArduino)**

**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** | **DATA0** | **DATA1-2** | **END** |
| **@** | **0x83** | **3** | **Index** | **Value** | **\*** |

**Request Integer Variable (AppYArduino)**

**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** | **DATA0** | **END** |
| **@** | **0x84** | **1** | **Index** | **\*** |

**Response Integer Variable Request (ArduinoYApp)**

**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** | **DATA0** | **DATA1-2** | **END** |
| **@** | **0x85** | **3** | **Index** | **Value** | **\*** |

**Sets Float Variable (AppYArduino)**

**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** | **DATA0** | **DATA1-4** | **END** |
| **@** | **0x86** | **5** | **Index** | **Value** | **\*** |

**Request Float Variable (AppYArduino)**

**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** | **DATA0** | **END** |
| **@** | **0x87** | **1** | **Index** | **\*** |

**Response Float Variable Request (ArduinoYApp)**

**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** | **DATA0** | **DATA1-4** | **END** |
| **@** | **0x88** | **5** | **Index** | **Value** | **\*** |

**Sets String Variable (AppYArduino)**

**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** | **DATA0** | **DATA1-M** | **END** |
| **@** | **0x89** | **M+1** | **Index** | **String** | **\*** |

**Request String Variable (AppYArduino)**

**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** | **DATA0** | **END** |
| **@** | **0x8A** | **1** | **Index** | **\*** |

**Response String Variable Request (ArduinoYApp)**

**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** | **DATA0** | **DATA1-M** | **END** |
| **@** | **0x8B** | **M+1** | **Index** | **String** | **\*** |