```
// Arduino BLE module identification and setup sketch
// Copyright, Arik Yavilevich

#include <SoftwareSerial.h>

/// Consts

// BLE module default pins
#define RX_PIN 8
#define TX_PIN 9
#define STATE_PIN 7
#define STATE_PIN_MISSING -1

// misc
#define SERIAL_BAUD 9600
#define SERIAL_TIMEOUT 60000 // 1 min
#define BLE_BAUD 9600 // for at mode and for data mode (CC41, HM-10 and MLT-BT05)
#define BLE_TIMEOUT 250 // 100 was ok for CC41 and HM-10 but not for MLT_BT05's AT+HELP command
#define INITIAL_DELAY 200

enum ModuleType { HM10, CC41, MLT_BT05, Unknown };
enum Operation {Quit, SetName, SetPass, SetStateBehavior, SetPower, DisplayMainSettings};

/// State
SoftwareSerial * ble = NULL; // supports modules conencted over software serial. no current support for Serial2, etc.
int rxPin, txPin, statePin;
ModuleType moduleType;

/// Special functions
// define here due to custom return type
ModuleType identifyDevice();
void doCommandAndEchoResult(const char * command, const __FlashStringHelper * meaning = NULL);
Operation getMenuSelection();

// Serial line end modes
// CR \r
// LF \n
// CR&LF \r\n

void setup()
{
    Serial.begin(SERIAL_BAUD);
    Serial.setTimeout(SERIAL_TIMEOUT);
    delay(INITIAL_DELAY);

    Serial.println(F("Arduino BLE module identification and setup sketch."));
    Serial.println(F("Interact with this interface using serial in CR&LF mode."));
    // Serial.println(F(""));
    do
    {
        do
        {
            openBLE();
        } while (determineConnectionState() == false);
    } while ((moduleType=identifyDevice()) == Unknown);
    displayMainSettings();
    do
```

```
    {
        Operation op = getMenuSelection();
        switch (op)
        {
        case SetName:
            setName();
            break;
        case SetPass:
            setPass();
            break;
        case SetStateBehavior:
            setStateBehavior();
            break;
        case SetPower:
            setPower();
            break;
        case DisplayMainSettings:
            displayMainSettings();
            break;
        default:
            Serial.println(F("Quitting. Sketch ended."));
            return;
        }
    } while (true);
}

/// BL functions

void openBLE()
{
    rxPin=readInt(F("Enter the number of the RX pin on the Arduino, TX on the module"), RX_PIN);
    txPin = readInt(F("Enter the number of the TX pin on the Arduino, RX on the module"), TX_PIN);
    statePin = readInt(F("Enter the number of the State pin on the Arduino, State on the module (enter -1 if not present or not connected)"), STATE_PIN);

    Serial.print(F("Opening serial connection to BLE module at pins: "));
    Serial.print(rxPin);
    Serial.print(F(", "));
    Serial.print(txPin);
    Serial.print(F(", "));
    Serial.print(statePin);
    Serial.println();

    // open and create object
    if (ble)
        delete ble;
    ble = new SoftwareSerial(rxPin, txPin);
    ble->begin(BLE_BAUD);
    ble->setTimeout(BLE_TIMEOUT);
    if(statePin!=STATE_PIN_MISSING)
        pinMode(statePin, INPUT);
}

bool determineConnectionState()
{
    if (statePin != STATE_PIN_MISSING)
    {
        Serial.println(F("Checking module state..."));
```

```
        // read state
        int state = digitalRead(statePin);
        // check if state is "blinking"
        unsigned long p1 = pulseIn(statePin, HIGH, 2000000);
        unsigned long p2 = pulseIn(statePin, LOW, 2000000);
        if (p1 == p2 && p1 == 0)
        {
            if (state == HIGH)
            {
                Serial.println(F("The signal on the state pin is HIGH. This means the device is connected and is in data mode. Sketch can't proceed."));
                return false;
            }
            else
                Serial.println(F("The signal on the state pin is LOW. This means the device is not connected and is in command mode."));
        }
        else // blinking
        {
            Serial.println(F("The signal on the state pin is \"blinking\". This usually means the device is not connected and is in command mode."));
            Serial.println(F("Consider reconfiguring the device to pass the state as-is. So it is simple to detect the state."));
        }
    }
    else
    {
        Serial.println(F("For this sketch, make sure the module is not connected to another BLE device."));
        Serial.println(F("This will make sure the device is in command mode."));
        Serial.println(F("A led on the module should be off or blinking."));
    }
    return true;
}

ModuleType identifyDevice()
{
    Serial.println(F("Detecting module type"));
    // check for HM-10
    ble->print("AT");
    String s = ble->readString();
    if (s.length() == 2 && s.compareTo("OK")==0)
    {
        Serial.println(F("HM-10 detected!"));
        return HM10;
    }
    else if (s.length() == 0)
    {
        // check for CC41
        ble->println("");
        s = ble->readString();
        if (s.length() == 4 && s.compareTo("OK\r\n") == 0)
        {
            Serial.println(F("CC41 detected!")); // An HM-10 clone
            return CC41;
        }
        else if (s.length() == 0)
        {
            // check for MLT_BT05
            ble->println("AT");
            s = ble->readString();
            if (s.length() == 4 && s.compareTo("OK\r\n") == 0)
```

```
            {
                Serial.println(F("MLT-BT05 detected!")); // An HM-10 clone
                return MLT_BT05;
            }
            else if (s.length() == 0)
            {
                Serial.println(F("No response received from module."));
                Serial.println(F("Verify that the module is powered. Is the led blinking?"));
                Serial.println(F("Check that pins are correctly connected and the right values were entered."));
                Serial.println(F("Are you using a logic voltage converter for a module that already has such logic on board?"));
                Serial.println(F("Are you using a module that expects 3.3v logic? You might need to do logic convertion on Arduio's TX pin (module's RX pin)."));
            }
            else
            {
                Serial.print(F("Unexpected result of length="));
                Serial.println(s.length());
                Serial.println(s);
            }
        }
        else
        {
            Serial.print(F("Unexpected result of length="));
            Serial.println(s.length());
            Serial.println(s);
        }
    }
    else
    {
        Serial.print(F("Unexpected result of length="));
        Serial.println(s.length());
        Serial.println(s);
    }
    return Unknown;
}

void displayMainSettings()
{
    if (moduleType == HM10)
    {
        doCommandAndEchoResult(("AT+HELP?"));
        doCommandAndEchoResult(("AT+VERS?"));
        doCommandAndEchoResult(("AT+NAME?"));
        doCommandAndEchoResult(("AT+PASS?"));
        doCommandAndEchoResult(("AT+ADDR?"));
        doCommandAndEchoResult(("AT+ROLE?"), F("Peripheral=0, Central=1"));
        doCommandAndEchoResult(("AT+POWE?"), F("0 = -23dbm, 1 = -6dbm, 2 = 0dbm, 3 = 6dbm"));
        doCommandAndEchoResult(("AT+MODE?"), F("Transmission Mode=0, PIO collection Mode=1, Remote Control Mode=2"));
        doCommandAndEchoResult(("AT+PIO1?"), F("Behavior of state pin, Blink on disconnect=0, Off on disconnect=1"));
    }
    else if (moduleType == CC41)
    {
        doCommandAndEchoResult(("AT+HELP"));
        doCommandAndEchoResult(("AT+VERSION"));
        doCommandAndEchoResult(("AT+NAME"));
        doCommandAndEchoResult(("AT+PASS"));
        doCommandAndEchoResult(("AT+ADDR"));
        doCommandAndEchoResult(("AT+ROLE"));
```

```
        doCommandAndEchoResult(("AT+POWE"), F("0 = -23dbm, 1 = -6dbm, 2 = 0dbm, 3 = 6dbm"));
    }
    else if (moduleType == MLT_BT05)
    {
        doCommandAndEchoResult(("AT+HELP"));
        doCommandAndEchoResult(("AT+VERSION"));
        doCommandAndEchoResult(("AT+NAME"));
        doCommandAndEchoResult(("AT+PIN"));
        doCommandAndEchoResult(("AT+LADDR"));
        doCommandAndEchoResult(("AT+ROLE"));
        doCommandAndEchoResult(("AT+POWE"), F("0 = -23dbm, 1 = -6dbm, 2 = 0dbm, 3 = 6dbm"));
    }
}

Operation getMenuSelection()
{
    Serial.println(F("0) Quit"));
    Serial.println(F("1) Set module name"));
    Serial.println(F("2) Set module password"));
    if(moduleType==HM10)
        Serial.println(F("3) Set module state pin behavior"));
    Serial.println(F("4) Set module power"));
    Serial.println(F("5) Display main settings"));
    int op = readInt(F("Enter menu selection"), 0);
    return (Operation)(op);
}

void setName()
{
    String val = readString(F("Enter new name"), F("HMSoft"));
    String command(F("AT+NAME"));
    command += val;
    doCommandAndEchoResult(command.c_str());
}

void setPass()
{
    String val = readString(F("Enter new pass/pin"), F("000000"));
    String command(moduleType ==MLT_BT05? F("AT+PIN") : F("AT+PASS"));
    command += val;
    doCommandAndEchoResult(command.c_str());
}

void setStateBehavior()
{
    String val = readString(F("Enter new state pin behavior 0 or 1"), F("1"));
    String command(F("AT+PIO1"));
    command += val;
    doCommandAndEchoResult(command.c_str());
    doCommandAndEchoResult("AT+RESET", F("to apply the AT+PIO1 command"));
}

void setPower()
{
    int dbm = readInt(F("Enter new module power (0 = -23dbm, 1 = -6dbm, 2 = 0dbm, 3 = 6dbm)"), 2); // 2 is the default
    String command(F("AT+POWE"));
    command += dbm;
```

```
    doCommandAndEchoResult(command.c_str());
}

/// Interface helper functions

int getLengthWithoutTerminator(String & s)
{
    int l = s.length();
    if (l == 0)
        return 0;
    if (s.charAt(l - 1) == '\r')
        return l - 1;
    return l;
}

int readInt(const __FlashStringHelper * message, int defaultValue)
{
    Serial.print(message);
    Serial.print(" [");
    Serial.print(defaultValue);
    Serial.println("] :");

    String s = Serial.readStringUntil('\n');
    if (getLengthWithoutTerminator(s) == 0)
        return defaultValue;
    return s.toInt();
}

String readString(const __FlashStringHelper * message, const __FlashStringHelper * defaultValue)
{
    Serial.print(message);
    Serial.print(" [");
    Serial.print(defaultValue);
    Serial.println("] :");

    String s = Serial.readStringUntil('\n');
    int l = getLengthWithoutTerminator(s);
    if (l == 0)
        return defaultValue;
    s.remove(l); // remove terminator
    return s;
}

void doCommandAndEchoResult(const char * command, const __FlashStringHelper * meaning)
{
    // announce command
    Serial.print(F("Sending command: "));
    Serial.print(command);
    if (meaning != NULL)
    {
        Serial.print(F(" ("));
        Serial.print(meaning);
        Serial.print(F(")"));
    }
    Serial.println();

    // send command
```

```
    if (moduleType == HM10)
        ble->print(command);
    else
        ble->println(command);

    // read and return response
    // don't use "readString", it can't handle long and slow responses (such as AT+HELP) well
    byte b;
    while (ble->readBytes(&b, 1) == 1) // use "readBytes" and not "read" due to the timeout support
        Serial.write(b);

    // normalize line end
    if (moduleType == HM10)
        Serial.println();
}

void loop()
{
}
```