Arduino Hardware and dependent libraries

Legoino

Disclaimer: LEGO® is a trademark of the LEGO Group of companies which does not sponsor, authorize or endorse this project.

Legoino is an Arduino Library for controlling all kinds of *LEGO* Powered UP devices. From the two port hub, Move Hub (e.g. Boost), DUPLO train hub, Technic Hub to several devices like distance and color sensor, tilt sensor, train motor, remote control, speedometer, etc. you can control almost everything with that library and your Arduino sketch.

It is also possible to use the "old" Power Function IR Modules and control them via an IR LED connected to a PIN of your ESP32 device. With the Hub emulation function you can even control an "old" Power Function Light or Motor with the Powered Up app.

The library is implemented for **ESP32** boards and uses the ESP32 NimBLE-Arduino library as dependency. This should be installed via the Arduino library manager before using Legoino.

Quickstart

You can find a step by step instruction to your first Legoino project on the following link: Quickstart Tutorial

Breaking Changes

Starting from version 1.0.0 many functions have been renamed and the global variables have been replaced by callback functions. In former versions the reading of sensor values of single or multiple sensors and even reading sensors from different hubs, was not working properly. Due to the changes to the NimBLE-Arduino library the callbacks can now be part of member functions and don't have to be defined globally.

So have a look at the changes and adapt your sketches to the new callbacks. You can find a migration guide here: Migration Guide

If you have questions regarding the migration of your sketches, don't hesitate to use the

gitter join chat

chat.

Usage Videos

In the following videos you can see with short examples what you can do with the library.

Remote control your Boost model example (just click the image to see the video)

Simple Train example (just click the image to see the video)

Simple Boost movement example (just click the image to see the video)

Simple Hub Emulation example as Bridge from PoweredUp to PowerFunction. With this you have an upgrade of your PowerFunction system and it works like a two Port PoweredUp hub.

Examples

All the included examples are a great source to find a solution or pattern for your problem you want to solve with your Arduino sketch.

You can find different examples in the "examples" folder. You can select the examples in your Arduino IDE via the Menu "File->Examples". Just have a look on the videos to see the examples running 😃

- **Boost.ino:** Example which uses the basic Boost movements (usable with M.T.R. 4 or Vernie model). http://www.youtube.com/watch?v=VgWObhyUmi0
- **ColorSensor.ino:** Example which reads in the color sensor value (attached to an arbitrary port) and uses the detected color to set the Hub LED accordingly. https://youtu.be/_xCd9Owy1nk
- **MoveHubDeviceInfo.ino:** Example which displays the various device infos (firmware version, battery level, RSSI, hardware version, tilt) in the serial monitor.
- **DistanceSensor.ino:** Example which reads in the input of the distance sensor and sets the Hub LED color dependent on the distance. https://youtu.be/TOAQtGGjZ6c
- **RotationSensor.ino:** Example which reads in the input of the Tacho motor angle to set the Hub LED dependent on the angle to the scale of rainbow colors. https://youtu.be/c3DHpX55uN0
- **TrainHub.ino:** Example for a PowererdUp Hub to set the speed of a train model. http://www.youtube.com/watch?v=o1hgZQz3go4
- **TrainColor.ino:** Example of PoweredUp Hub combined with color sensor to control the speed of the train

dependent on the detected color. https://youtu.be/GZ0fqe3-Bhw
- **HubEmulation.ino:** Example of an emulated PoweredUp Hub two port hub (train hub) which could receive signals from the PoweredUp app and will send out the signals as IR commands to a Powerfunction remote receiver. https://www.youtube.com/watch?v=RTNexxT4-yQ
- **PoweredUpRemoteAutoDetection.ino:** Example of connection of PoweredUp and PoweredUpRemote where the device type is fetched automatically and the order in which you switched on the hubs is no longer relevant.
- **ControlPlusHub.ino:** Example with connection of ControlPlusHub (TechnicHub) where a Tacho Motor on Port D is controlled.
- **Mario.ino** Example of connection to a Mario Hub to read in sensor notifications about the Barcode/Tag sensor, Color sensor, Pants sensor and Gesture sensor.

Setup and Usage

Just install the library via the Arduino Library Manager.

First example

Just have a look on the Quickstart Tutorial.

Connection procedure

To setup a connection to your hub, the Hub instance has to be initialized. This will trigger a scan procedure to look if a LEGO Hub is active. If the library found an active hub, it will read out its data (Hub Address, Hub Name, etc.) and changes the state to `myHub.isConnecting() == true`

Now you are ready to connect to the hub with the command `myHub.connectHub();`

If the library changes the state to `myHub.isConnected() == true` you are ready to go and do some cool stuff 😁

In the `setup` part of your Arduino sketch, just initialize your Hub:

In the main `loop` just add the following connection flow:

```
if (myHub.isConnecting()) {
  myHub.connectHub();
  if (myHub.isConnected()) {
    Serial.println("We are now connected to the HUB");
  } else {
    Serial.println("We have failed to connect to the HUB");
  }
}
```

Motor Commands

There are different types of motors in the *LEGO* ecosystem. The Basic Motor (e.g. Train Hub), The Tacho motor and the absolute motor. The prefixes `Basic`, `Tacho` and `Absolute` tell you which command fits for what type of motor. The Tacho Motor commands can also be used with the Absolute motor.

```
void stopBasicMotor(byte port);
void setBasicMotorSpeed(byte port, int speed);

void setAccelerationProfile(byte port, int16_t time);
void setDecelerationProfile(byte port, int16_t time);
void stopTachoMotor(byte port);
void setTachoMotorSpeed(byte port, int speed, byte maxPower = 100, BrakingStyle brakingSty
void setTachoMotorSpeedForTime(byte port, int speed, int16_t time, byte maxPower = 100, Br
void setTachoMotorSpeedForDegrees(byte port, int speed, int32_t degrees, byte maxPower = 1
void setTachoMotorSpeedsForDegrees(int speedLeft, int speedRight, int32_t degrees, byte ma

void setAbsoluteMotorPosition(byte port, int speed, int32_t position, byte maxPower = 100,
void setAbsoluteMotorEncoderPosition(byte port, int32_t position);
```

Hub Commands

For some use cases it is required to get the hub address (check a specific hub), the hub type (check a hub type) or the hub name. It is also possible to change the hub name and shutdown the hub.

```
NimBLEAddress getHubAddress();
HubType getHubType();
std::string getHubName();
void setHubName(char name[]);
void shutDownHub();
```

LED Commands

To control the Hub LEDs, you can use predefined color Variables in the `Color` enum, RGB values or HSV values with the following commands:

```
void setLedColor(Color color);
void setLedRGBColor(char red, char green, char blue);
void setLedHSVColor(int hue, double saturation, double value);
```

Sensor and hub property handling

To get notified about sensor value updates (Button, Hub properties like voltage, RSSI, Tacho motor encoder, Speedometer, color sensor, distance sensor, ...), callback functions are used. After you read the following section you can also have a look into the examples which are included in the library. They are always a good source to find solutions/patterns for problems you want to solve.

Callbacks

To use the callbacks you have to do the following steps. Part of the callback ist the reference of the hub instance. So you can use the hub instance which has triggered the callback function to control attached motors or devices.

Write callback function for port devices

To read in changes of devices which are attached to a port (built-in or external), you have to write a function with the following signature:

```
typedef void (*PortValueChangeCallback)(void *hub, byte portNumber, DeviceType deviceType, u
```

Example:

```
// callback function to handle updates of sensor values
void tachoMotorCallback(void *hub, byte portNumber, DeviceType deviceType, uint8_t *pData)
{
  Lpf2Hub *myHub = (Lpf2Hub *)hub;

  Serial.print("sensorMessage callback for port: ");
  Serial.println(portNumber, DEC);
  if (deviceType == DeviceType::MEDIUM_LINEAR_MOTOR)
  {
    int rotation = myHub->parseTachoMotor(pData);
    Serial.print("Rotation: ");
    Serial.print(rotation, DEC);
    Serial.println(" [degrees]");
    myHub->setLedHSVColor(abs(rotation), 1.0, 1.0);
  }
}
```

This function will be called when a value update appears and you can react on the new value. In this case the LED color changes dependent on the motor rotation.

Write callback function for hub properties

To read in changes of hub properties (button, RSSI, battery level, ...), you have to write a function with the following signature:

```
typedef void (*HubPropertyChangeCallback)(void *hub, HubPropertyReference hubProperty, uint8
```

Example:

```
// callback function to handle updates of hub properties
void hubPropertyChangeCallback(void *hub, HubPropertyReference hubProperty, uint8_t *pData)
{
  Lpf2Hub *myHub = (Lpf2Hub *)hub;
  Serial.print("HubAddress: ");
  Serial.println(myHub->getHubAddress().toString().c_str());

  Serial.print("HubProperty: ");
  Serial.println((byte)hubProperty, HEX);

  if (hubProperty == HubPropertyReference::RSSI)
  {
    Serial.print("RSSI: ");
    Serial.println(myHub->parseRssi(pData), DEC);
    return;
  }

  if (hubProperty == HubPropertyReference::BATTERY_VOLTAGE)
  {
    Serial.print("BatteryLevel: ");
    Serial.println(myHub->parseBatteryLevel(pData), DEC);
    return;
  }

  if (hubProperty == HubPropertyReference::BUTTON)
  {
    Serial.print("Button: ");
    Serial.println((byte)myHub->parseHubButton(pData), HEX);
    return;
  }
}
```

This function will be called when a value update appears and you can react on the new value. In this case it will check the hub button state (pressed, released) or the RSSI value has changed or the battery level has changed.

Activate notifications

To get calls to your callback functions you have to register or activate it for the properties you want to get informed about updates. This has to be done after the hub is connected (not before). If you want to register updates for several properties/sensors, just add a short delay (50-100ms) after each register/activate call.

For Port related updates you have to use the function

```
  void activatePortDevice(byte portNumber, byte deviceType, PortValueChangeCallback portValu
```

Example:

```
// get notified for value updates of the device which is connected on port D
myMoveHub.activatePortDevice(portD, tachoMotorCallback);
delay(50);
myMoveHub.activateHubPropertyUpdate(HubPropertyReference::BUTTON, buttonCallback);
```

For Hub property related updates you have to use the function

```
  void activateHubPropertyUpdate(HubPropertyReference hubProperty, HubPropertyChangeCallback
```

Example:

```
// get notified for value updates of the build in Button of the hub
myMoveHub.activateHubPropertyUpdate(HubPropertyReference::BUTTON, buttonCallback);
```

You can also check if an expected device is really connected to a port by using the newly introduced function `checkPortForDevice` like in the following example:

```
Serial.print("check ports... if needed sensor is already connected: ");
byte portForDevice = myHub.getPortForDeviceType((byte)DeviceType::COLOR_DISTANCE_SENSOR);
Serial.println(portForDevice, DEC);
// check for expected port number where the device should be connected
if (portForDevice == 1)
{
        Serial.println("activatePortDevice");
        myHub.activatePortDevice(portB, colorDistanceSensorCallback);
}
```

Hub emulation

The Hub emulation feature is in *BETA* mode. You can test it and if you find any issues or needed new requirements, just open an issue in github project

The basic idea is that the ESP32 controller acts as a hub and could be controlled via the PoweredUp App.

First you have to create an instance of a hub where you can define the name of the Hub which will be advertised and the type of the hub. In the current implementation only the `POWERED_UP_HUB` type is supported and only a connected `TRAIN_MOTOR` and `HUB_LED` will work.

```
#include "Lpf2HubEmulation.h"
#include "LegoinoCommon.h"

// create a hub instance
Lpf2HubEmulation myEmulatedHub("TrainHub", HubType::POWERED_UP_HUB);
```

To get notified if a command is sent to the app, a callback has to be defined and registered with the `setWritePortCallback`. If the app is connected and send out a command, you can define what you want to do if you receive that command. For example you can send out a power function command to act as a hub relay.

```
  // define the callback function if a write message event on the characteristic occurs
  myEmulatedHub.setWritePortCallback(&writeValueCallback);
  myEmulatedHub.start();


void writeValueCallback(byte port, byte value)
{
  Serial.println("writeValueCallback: ");
  Serial.println(port, HEX);
  Serial.println(value, HEX);

  if (port == 0x00)
  {
    //do something when port 0x00 (A) has received a value
  }

  if (port == 0x01)
  {
    //do something when port 0x01 (B) has received a value
  }

  if (port == 0x32)
  {
    //do something when port 0x32 (LED) has received a value
  }
}
```

To signalize the app which devices are connected you have to send some commands with the `attachDevice` method. You can define on which port, which device type is connected.

```
  // if an app is connected, attach some devices on the ports to signalize
  // the app that values could be received/written to that ports
  if (myEmulatedHub.isConnected && !myEmulatedHub.isPortInitialized)
  {
    delay(1000);
    myEmulatedHub.isPortInitialized = true;
    myEmulatedHub.attachDevice(0x00, DeviceType::TRAIN_MOTOR);
    delay(1000);
    myEmulatedHub.attachDevice(0x32, DeviceType::HUB_LED);
    delay(1000);
    myEmulatedHub.attachDevice(0x01, DeviceType::TRAIN_MOTOR);
    delay(1000);
  }
```

PowerFunction IR

To use the PowerFunction Infrared library you have to connect a IR-LED to your controller. The PowerFunction part of the Legoino library will work on different hardware architectures (AVR, ESP8266, ESP32). To instantiate a new object, you have to define on which pin the IR LED is connected. Additionally you can define the Power function channel which should be used. Pay attention, that the channel in the library starts with 0 and on the physical IR Lego controller it starts with 1. So if you want to control channel 1 of your physical device, you have to use channel 0 in the PowerFunction part of the library.

```
#include "PowerFunctions.h"
// create a power functions instance (IR LED on Pin 12, IR Channel 0)
PowerFunctions pf(12, 0);
```

Then you can control the output ports (Red, Blue) with the following possible commands. The ports could be defined with the `PowerFunctionsPort` enum. For the PWM values the `PowerFunctionsPwm` enum is used.

```
  void single_pwm(PowerFunctionsPort port, PowerFunctionsPwm pwm);
  void single_pwm(PowerFunctionsPort port, PowerFunctionsPwm pwm, uint8_t channel);
  void single_increment(PowerFunctionsPort port);
  void single_increment(PowerFunctionsPort port, uint8_t channel);
  void single_decrement(PowerFunctionsPort port);
  void single_decrement(PowerFunctionsPort port, uint8_t channel);
  void combo_pwm(PowerFunctionsPwm redPwm, PowerFunctionsPwm bluePwm);
  void combo_pwm(PowerFunctionsPwm redPwm, PowerFunctionsPwm bluePwm, uint8_t channel);
```

There is a helper function to convert speed values from `-100...100` to the discrete PWM values:

```
PowerFunctionsPwm speedToPwm(byte speed);
```

Boost

The Boost functions are a higher level of abstraction for moving the Vernie or M.T.R. 4 model one step forward/back and rotate the model or move with an arc.

Add the following include in your *.ino sketch:

```
#include "Boost.h"
Boost myBoostHub;
```

Basic movements (Vernie, M.T.R. 4)

If you want to move Vernie or M.T.R. 4 you can use the following commands. These commands are using the underlying basic motor commands and are adjusted to the Boost grid map.

If you want to move forward for a specific number of steps, just use the following command:

```
myBoostHub.moveForward(1) // move one step forward
myBoostHub.moveForward(3) // move three steps forward
```

If you want to move back for a specific number of steps, just use the following command:

```
myBoostHub.moveBack(1) // move one step back
myBoostHub.moveBack(3) // move three steps back
```

If you want to rotate for a specific angle, just use the following commands:

```
myBoostHub.rotateLeft(90) // rotate 90 degrees left
myBoostHub.rotateRight(180) // rotate 180 degrees right
myBoostHub.rotate(360) // rotate 360 degrees right (positive angles means right, negative me
myBoostHub.rotate(-180) // rotate 180 degrees left (positive angles means right, negative me
```

If you want to move with an arc for a specific angle, just use the following commands:

```
myBoostHub.moveArcLeft(180) // move with an arc for 180 degrees to the left
myBoostHub.moveArcRight(90) // move with an arc for 90 degrees to the right
myBoostHub.moveArc(270) // move with an arc for 270 degrees to the right (positive angles me
myBoostHub.moveArc(-90) // move with an arc for 90 degrees to the left (positive angles mean
```

Mario Hub (#71360)

With Legoino it is possible to connect to the Mario Hub and read out sensor values from the

- Pant Sensor
- Color and Tag/Barcode Sensor
- Gesture Sensor
- Voltage Sensor

You can do this via the standard "sensor notification" procedure. Just have a look into the **Mario.ino** example sketch.

There is an undocumented hub property `0x12` to control the volume of the hub. This feature can be used with the Legoino function `setMarioVolume(volume)` with a volume value from 0..100 in %.

Connection to more than 3 hubs

It is possible to connect to up to 9 hubs in parallel with a common ESP32 board. To enable the connection to more than 3 hubs, you have to change a single configuration of the NimBLE library. Just open the `nimconfig.h` file located in your Arduino library folder in the directory `NimBLE-Arduino/src`. Open the file with an editor and change the following settings to your demands:

```
/** @brief Sets the number of simultaneous connections (esp controller max is 9) */
#define CONFIG_BT_NIMBLE_MAX_CONNECTIONS 3
```

Then close the Arduino environment and open it again to force the rebuild of the library. Open your sketch build and upload it and be happy with multiple connections.

Debug Messages

The standard `log_d`, `log_w`, `log_xx` messages are used. The log levels could be set via the Arduino environment and the messages are sent to the serial monitor.

Credits

Hands up to Lego, that they have recently open-sourced the Specification https://github.com/LEGO/lego-ble-wireless-protocol-docs

Thanks to @JorgePe and all contributors for the reverse engineering part https://github.com/JorgePe/BOOSTreveng

Thanks to @jakorten for his SWIFT iOS App https://github.com/jakorten/UpControl

Thanks @nathankellenicki for his brilliant structured node module https://github.com/nathankellenicki/node-poweredup

Thanks to @h2zero for developing a new BLE library based on the NimBLE project and supporting Legoino with the possibility in changing the callback functions that it works also for member functions.

Thanks to @giancann, @marcrupprath and @albant for the hub emulation idea and first implementation.

Thanks to @fvanroie for his contribution about callbacks.

Thanks for the original PowerFunctions Library by @jurriaan

Thanks to @wmarkow for his detailed input about hub emulation issues on Android devices and his proposals.

Remarks

Prerequisite of that library is the NimBLE-Arduino library (https://github.com/h2zero/NimBLE-Arduino) with at least version 1.0.1. Otherwise the notifications of changed characteristic values will not work. So just install the version 1.0.1 of that library via the Arduino Library manager or the platform.io library manager (https://www.arduinolibraries.info/libraries/nim-ble-arduino) as a prerequisite.

Up to now the library is only tested for a Powered Up Train controllers, Boost controllers, Control+ Hubs, PoweredUp Remote, Duplo Train Hub and Mario Hub. You can connect to your Hub, set the LED color, set the Hub name, control the motors (speed, port, movements) and shut down the Hub via a Arduino command. You also are able to read in hub device infos (RSSI, battery level, tilt) and sensor values (color, distance, rotation angle).

ToDo

- Virtual Ports
- HW Families