Getting Started Guide
- duinoPRO UNO -
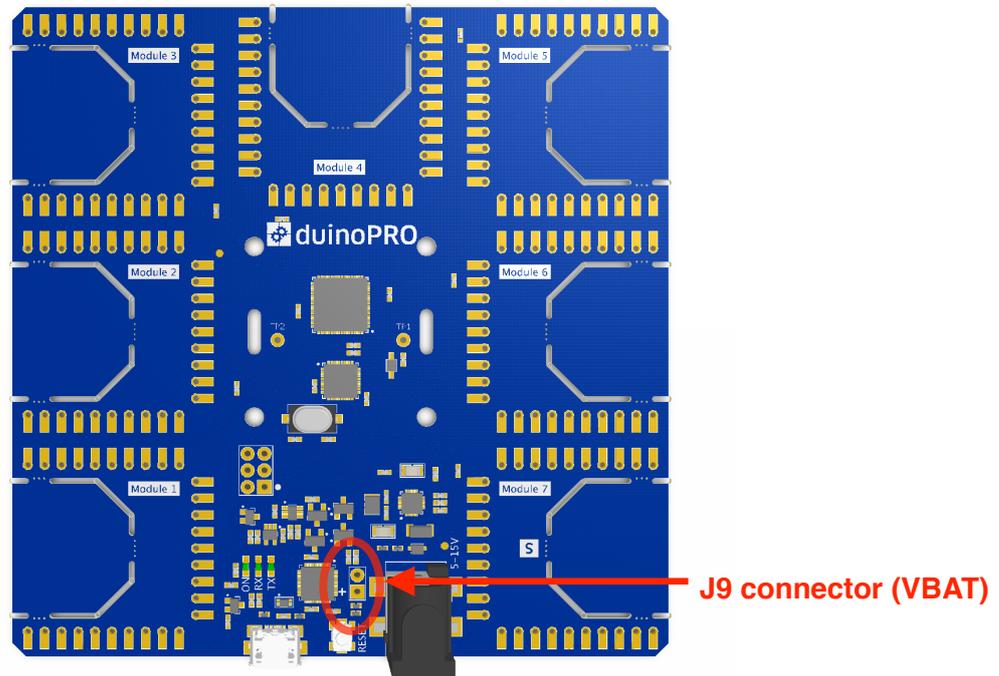

Figure 1. duinoPRO UNO baseboard - 3D render (front)

Installing duinoPRO support in the Arduino™ IDE

Follow the instructions at https://github.com/duinoPRO/install/blob/master/README.md to install support for duinoPRO in your Arduino™ IDE or install the Arduino™ IDE from scratch.

Setting up the hardware

Mounting modules

duinoPRO is built on the concept of using modules that can be mounted on a baseboard in multiple ways depending on how you plan to use the hardware.

During development, test or while playing with duinoPRO, it is useful to be able to the move modules around between module spaces. To do this, you can solder standard 0.1" sockets onto module spaces on your baseboard and standard 0.1" header onto your modules.

Once you are ready for your duinoPRO design to become more permanent it is possible to solder modules into place directly on your baseboard. The duinoPRO baseboard has snap-out "mouse bites" on all module spaces. For modules featuring a through-hole connector, you will need to snap these out in order to solder the module into place.

Serial modules **S** - Module space 7

The duinoPRO UNO microprocessor has a single serial port that is shared between the USB port and the Module 7 space. Modules requiring a serial port connection are identified by the serial port symbol, **S** ,printed on the bottom side. On the duinoPRO UNO baseboard, these can only be fitted to the Module 7 space (also marked with the serial port symbol, **S**). **Make sure that you fit any modules requiring a serial connection to Module 7 on your duinoPRO UNO baseboard, especially if you are soldering them permanently into place!**

Powering the baseboard

The duinoPRO UNO baseboard can be powered in a number of different ways.
- via the micro USB port (also used for programming)
- via the DC barrel connector (input must be between 5 and 15V and supply enough current to power the modules you have chosen)
- via the J9 battery connector (*VBAT* signal) (input must be between 3 and 15V and supply enough current to power the modules you have chosen)
- via a yellow duinoPRO power module e.g. a **Coin Cell** module (*VBAT* signal)

Programming the baseboard

To program your duinoPRO UNO baseboard, connect your computer to the baseboard using a micro USB cable. Select *duinoPRO UNO* from the drop-down menu under *Tools / Board* in the Arduino™ IDE, and the USB port that your board is plugged in on from the *Tools / Port* drop-down menu. To upload a sketch to the board, click on the forward arrow button, ⊕, in the IDE menu. For those already experienced with Arduino™, this should all be very familiar.

Accessing Baseboard Facilities

Before starting on specific modules, the duinoPRO UNO baseboard contains a small number of shared facilities that you may wish to access from your sketch. These include:
- the ability to measure battery voltage
- a baseboard "ON" LED to indicate that the board is powered
- serial port control

To use these facilities, you need to include the duinoPRO header file and declare an object of type `duinoPRO` at the top of your sketch:

```
#include <duinoPRO.h>

duinoPRO myBaseboard;
```

Baseboard "ON" LED

To control the baseboard LED:

```
myBaseboard.setLed(true);      // Turns the LED on
myBaseboard.setLed(false);     // Turns the LED off
```

Note that turning the baseboard "ON" LED off lowers the power consumption of the baseboard.

Measuring Battery Voltage

VBAT is a signal on the baseboard and on all of the module spaces. If the board is powered from *VBAT* via a yellow power supply module such as the duinoPRO **Coin Cell** Module or directly via the baseboard connector *J9* (marked in *figure 1* above, and not fitted by default), the voltage level of this signal can be measured in software for battery monitoring purposes.

Measuring battery voltage is a two step process. First the battery sense circuit needs to be enabled, using the `enableVbatSense()` method. Then the battery voltage can be read using `getVbat()`. Optionally the battery sense circuit can be disabled again using `disableVbatSense()`. This may be desirable as disabling the battery voltage measurement lowers the power consumption of the baseboard.

For example:

```
float myBattVoltage;

myBaseboard.enableVbatSense();
myBattVoltage = myBaseboard.getVbat();
myBaseboard.disableVbatSense();
```

Serial Port Control

As explained above, the duinoPRO UNO microprocessor has a single serial port that is shared between the USB port and the Module 7 space. The serial port is used, when a sketch is running, either:
  a) For sending messages over the USB port (useful for debugging), or
  b) For communicating with a module that requires a serial port connection (e.g. RS232, Bluetooth LE, Sigfox). Modules requiring a serial port connection are identified by the serial port symbol, **S** ,printed on the bottom side. On the duinoPRO UNO baseboard, these can only be fitted to the Module 7 space (also marked with the serial port symbol, **S**).

The following code snippet shows how to switch the duinoPRO UNO's serial port between these two modes:

```
myBaseboard.serialDebugMode();      // Puts serial port into
                                    // debug mode (a) above)
myBaseboard.serialModuleMode();     // Puts serial port into
                                    // module mode (b) above)
```

Note that when a sketch starts up, the duinoPRO UNO serial port is by default connected to the USB port for debug.

Remember to add the command `Serial.begin(baud_rate);` to the `setup()` function in your sketch as per regular Arduino™ requirements, where `baud_rate` is your chosen serial baud rate. See Arduino™ instructions for more details: https://www.arduino.cc/en/Reference/Serial.

You can use `Serial.print()` to test writing debug messages to the serial monitor accessible at *Tools | Serial Monitor* in the Arduino™ IDE.

Modules

Adding module support

To add support for a particular type of module to your sketch, include the relevant library under *Sketch | Include Library* in the Arduino™ IDE. The required header file should appear at the top of your sketch.

Declaring modules

At the start of a sketch, all hardware modules that you wish to access on your duinoPRO baseboard need to be declared. This declaration takes a single parameter, which is the location of the module on the baseboard. Module locations are marked on the baseboard e.g. "Module 1", "Module 2" etc.

For example:

```
dP_LedButton myLedButton(2);        // LED/Button Module in
                                    // space 2
dP_MicroSD mySdCard(3);             // Micro SD Module in
                                    // space 3
dP_LedButton myLedButton2(4);       // 2nd LED/Button Module
                                    // in space 4
```

Note that it is often possible to have more than one module of the same type fitted to a baseboard.

Note that this differs from Arduino™ - typically libraries for Arduino™ shields do not require the user to declare them explicitly in the sketch. The library handles this for you. duinoPRO differs in this regard for two reasons. Firstly, the software needs to

know the location of the module. Secondly, this allows for simple implementation when an application requires more than one module of the same type, something which is not usually possible with a shield arrangement.

Initialising modules

As per Arduino™ convention, duinoPRO module classes have a `begin()` method which must be called in the `setup()` function. The `begin()` methods for different modules take different arguments. Some may take no arguments at all. See the header file for the relevant module for more information.

Reading and writing pins

The interface between the duinoPRO baseboard and each module includes 6 general purpose input/output (GPIO) signals that can be directly controlled from the sketch (see *figure 2* at the end of this guide for the module interface definition). For most types of modules, these are used for specific purposes related to the module, and are controlled from within the library, so the user doesn't need to control them directly.

If you want to read and write pins directly, for example to control or sense external hardware, typically from a hardware perspective you would do this through a hardware breakout module (e.g. the **IO Breakout** module).

Or you may want to read and write pins directly because you're writing or modifying a module library, or even because you've just wired something directly to the baseboard. The code you need is the same in all cases.

Firstly, as always, you need to include the relevant library and declare the module and tell the software where it is.

For example:

```
#include <dP_Module.h>

dP_Module myModule(2);
```

Once a module is declared, its pins can be accessed using the module name and pin number. For example, to read pin 3 of a module declared immediately above, you would write:

```
myModule.pin(3).read();
```

Rather than referring to a pin as `pin(3)`, it can be preferable to give pins a recognisable name. This is done using the following code (**note** that you need to include `dP_Module.h` to access the `dP_Pin` class):

```
        dP_Pin myPin = myModule.pin(3);
```
Then, to read the pin, you would write:

```
        myPin.read();
```

Note that if you want your `dP_Pin` object to be accessible from within both the `setup()` and `loop()` functions, you will need to ensure that it is in scope by declaring it above `setup()`.

Note that the pins on any module type can be accessed like this. For most types of modules, however, you would not directly access GPIO pins, but use the methods specific to that module type. You may however want to access pins directly if you're adding extra functionality to the module's library (or writing a library from scratch, of course).

Setting Pin Direction

The direction of a pin (input or output) is set using the `mode()` method, e.g.:

```
        myPin.mode(OUTPUT);         // sets myPin as an output
        myOtherPin.mode(INPUT);     // sets myOtherPin as an input
```

How this differs from Arduino™ - Arduino™ allow inputs to optionally be pulled up. This is not available on duinoPRO UNO although it is likely to be available on future baseboards.

Reading and Writing to/from a Pin

Once a pin has been configured as an output, it can be written as follows:

```
        myPin.write(HIGH);          // sets myPin high
        myPin.write(LOW);           // sets myPin low
```

Similarly, an input pin can be read as follows:

```
        bool readValue;
        readValue = myOtherPin.read();
```

How this differs from Arduino™ - The syntax here is different from Arduino™. `read()`, `write()` and `mode()` here are methods on the `dP_Pin` class, whereas for Arduino™ the equivalent functions are `digitalRead()`, `digitalWrite()` and `pinMode()`. **Note** that these are functions rather than methods and take the pin as an argument instead. For consistency with the rest of the API, duinoPRO uses methods.

Writing your first sketch

As an example, let's write a sketch that uses a duinoPRO **LED/Button** module, and implements a simple program that controls an LED using a button.

Steps:
1.   Include the libraries related to your modules using *Sketch | Include Libraries* from the Arduino™ IDE menu bar. Also include `duinoPRO.h` if you need it, which is often the case.
2.   Declare the modules that you're using.
3.   Typically modules require a call to a `begin()` method. Add this to the `setup()` function in your sketch.
4.   Write your application. In this case:

```cpp
#include <dP_LedButton.h>


//declare duinoPRO LED/Button module instance (module mounted in
space 2)
dP_LedButton myLedButton(2);


void setup() {
  // put your setup code here, to run once:
  myLedButton.begin();
}

void loop() {
  // put your main code here, to run repeatedly:
  // read button 1
  if (myLedButton.readButton(1))
  {
    //if button 1 set, set LED 1 to green
    myLedButton.setLed(1, dP_LedButton::GREEN);
  }
  else
  {
    //else, turn LED 1 off
    myLedButton.setLed(1, dP_LedButton::OFF);
  }
}
```

Note that a modified version of this example sketch can be found at https://github.com/duinoPRO/firmware/blob/master/libraries/dP_LedButton/ examples/LedButton-Button/LedButton-Button.ino.

EXTIO4

EXTIO3

EXTIO2

GND

GPIO 1/INT

SPI CS

RESERVED

MCU RESET

SCL2

SDA2

EXTIO1

GND

ADDRESS 3

ADDRESS 2

ADDRESS 1

ADDRESS 0

GPIO 2/USART CTS

GPIO 3/USART RTS

GPIO 4/USART RX

GPIO 5/USART TX
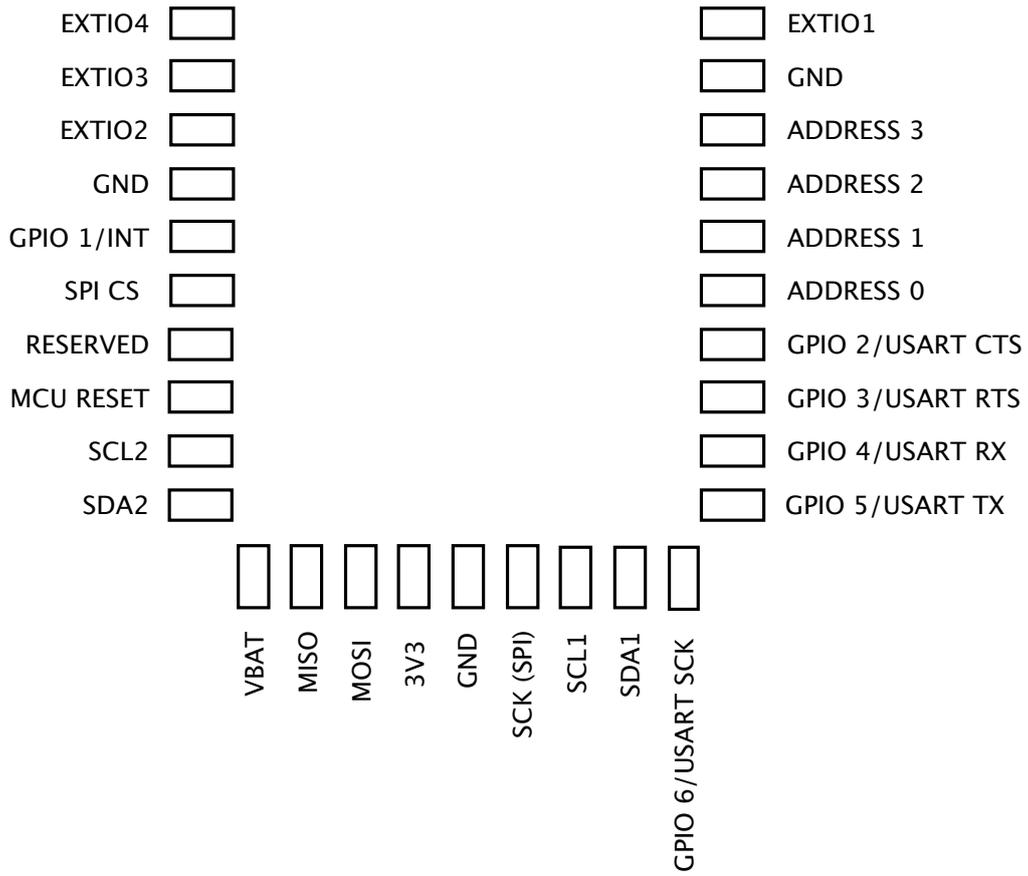
VBAT

MISO

MOSI

3V3

GND

SCK (SPI)

SCL1

SDA1

GPIO 6/USART SCK

Figure 2. duinoPRO module interface definition