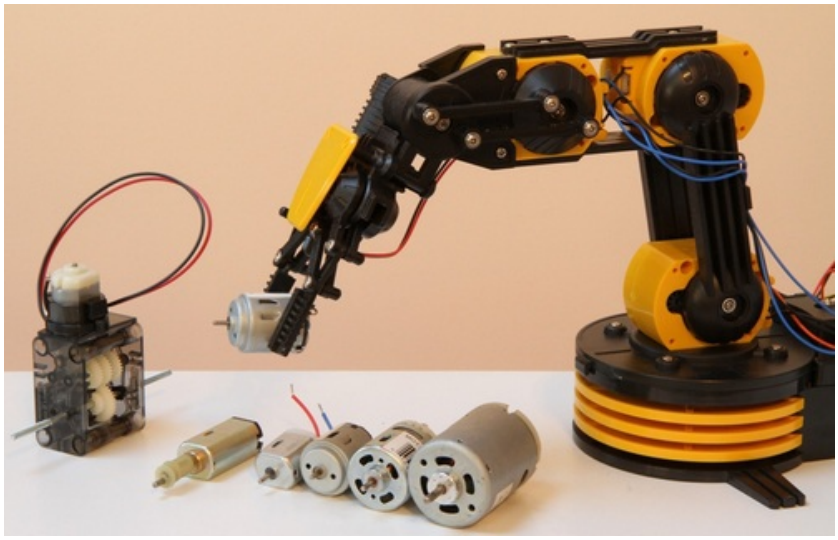
adafruit learning system

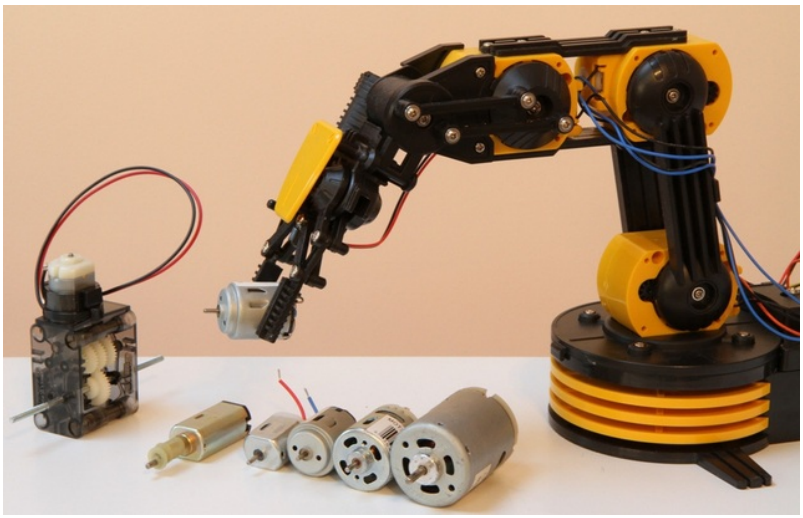
AFMotor Library Reference

Created by Bill Earl



Guide Contents

AF_DCMotor Class



The AF_DCMotor class provides speed and direction control for up to four DC motors when used with the Adafruit Motor Shield. To use this in a sketch you must first add the following line at the beginning of your sketch:

```
#include <AFMotor.h>
```

AF_DCMotor *motorname*(*portnum*, *freq*)

> This is the constructor for a DC motor. Call this constructor once for each motor in your sketch. Each motor instance must have a different name as in the example below.

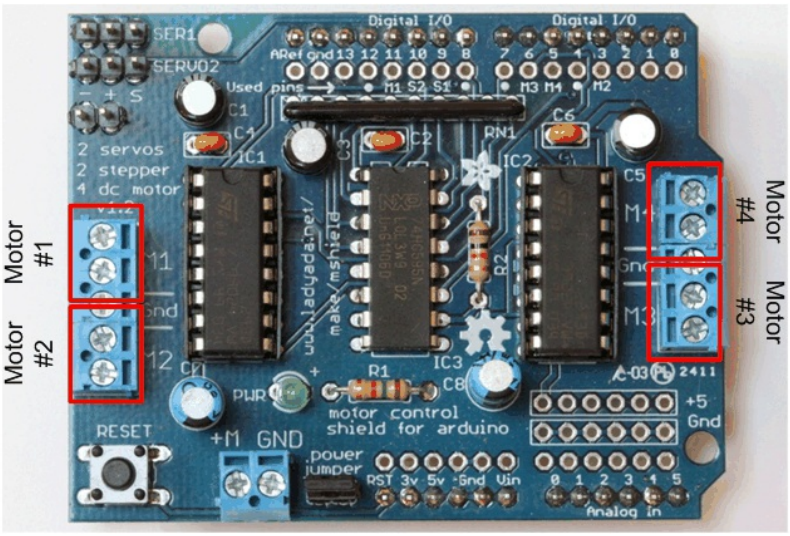
Parameters:

- **port num** - selects which channel (1-4) of the motor controller the motor will be connected to
- **freq** - selects the PWM frequency. If no frequency is specified, 1KHz is used by default.

> Frequencies for channel 1 & 2 are:
>
> - MOTOR12_64KHZ
> - MOTOR12_8KHZ
> - MOTOR12_2KHZ
> - MOTOR12_1KHZ
>
> Frequencies for channel 3 & 4 are:
>
> - MOTOR34_64KHZ
> - MOTOR34_8KHZ
> - MOTOR34_1KHZ

Example:

```
AF_DCMotor motor4(4); // define motor on channel 4 with 1KHz default PWM
AF_DCMotor left_motor(1, MOTOR12_64KHZ);  // define motor on channel 1 with 64KHz PWM
```



Note: *Higher frequencies will produce less audible hum in operation, but may result in lower torque with some motors.*

setSpeed(*speed*)

Sets the speed of the motor.

Parameters:

- **speed**- Valid values for 'speed' are between 0 and 255 with 0 being off and 255 as full throttle.

Example:

```
motor1.setSpeed(255);  // Set motor 1 to maximum speed
motor4.setSpeed(127);  // Set motor 4 to half speed
```

***Note*:** *DC Motor response is not typically linear, and so the actual RPM will not necessarily be proportional to the programmed speed.*

run(*cmd*)

Sets the run-mode of the motor.

Parameters:

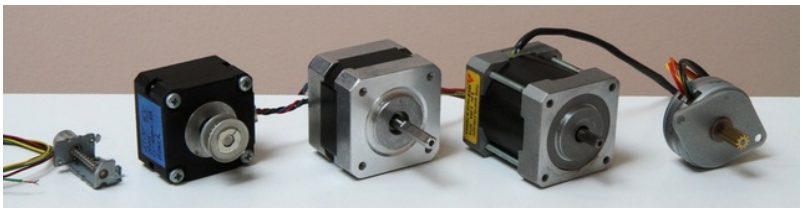
- **cmd** - the desired run mode for the motor

> Valid values for cmd are:
>
> - **FORWARD** - run forward (actual direction of rotation will depend on motor wiring)
> - **BACKWARD** - run backwards (rotation will be in the opposite direction from FORWARD)
> - **RELEASE** - Stop the motor. This removes power from the motor and is equivalent to setSpeed(0). The motor shield does not implement dynamic breaking, so the motor may take some time to spin down

Example:

```
motor.run(FORWARD);
delay(1000);  // run forward for 1 second
motor.run(RELEASE);
delay(100);  // 'coast' for 1/10 second
motor.run(BACKWARDS);  // run in reverse
```

AF_Stepper Class



The AF_Stepper class provides single and multi-step control for up to 2 stepper motors when used with the Adafruit Motor Shield. To use this in a sketch you must first add the following line at the beginning of your sketch:

```
#include <AFMotor.h>
```

AF_Stepper *steppername*(*steps*, *portnumber*)

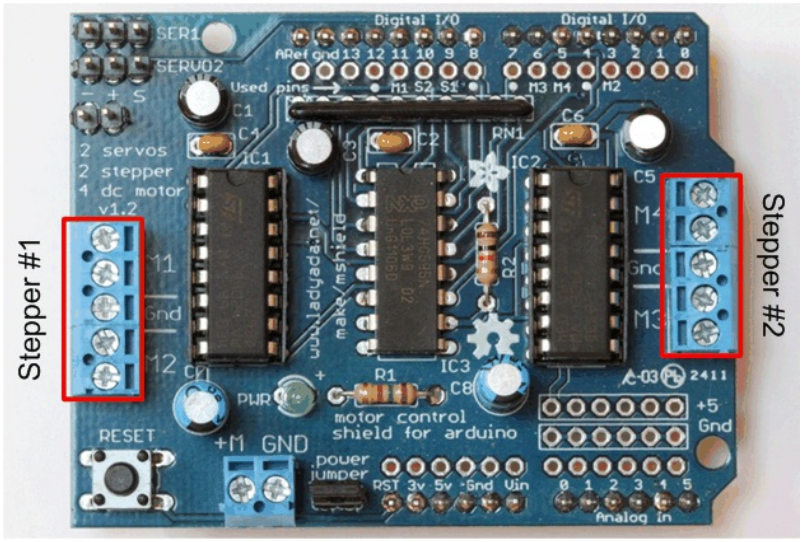
The AF_Stepper constructor defines a stepper motor. Call this once for each stepper motor in your sketch. Each stepper motor instance must have a unique name as in the example below.

Parameters:

- **steps** - declare the number of steps per revolution for your motor.
- **num** - declare how the motor will be wired to the shield.

> Valid values for 'num' are 1 (channels 1 & 2) and 2 (channels 3 & 4).

Example:



```
AF_Stepper Stepper1(48, 1);  // A 48-step-per-revolution motor on channels 1 & 2
AF_Stepper Stepper2(200, 2);   // A 200-step-per-revolution motor on channels 3 & 4
```

step(*steps, direction, style*)

Step the motor.

Parameters:

- **steps** - the number of steps to turn
- **direction** - the direction of rotation (**FORWARD** or **BACKWARD**)
- **style** - the style of stepping:

> Valid values for 'style' are:
>
> - **SINGLE** - One coil is energized at a time.
> - **DOUBLE** - Two coils are energized at a time for more torque.
> - **INTERLEAVE** - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
> - **MICROSTEP** - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

Note: *Step is a synchronous command and will not return until all steps have completed. For concurrent motion of two motors, you must handle the step timing for both motors and use the "onestep()" function below.*

Example:

```
Stepper1.step(100, FORWARD, DOUBLE); // 100 steps forward using double coil stepping
Stepper2.step(100, BACKWARD, MICROSTEP);   // 100 steps backward using double microst
epping
```

setSpeed(*RPMspeed*)

set the speed of the motor

Parameters:

- Speed - the speed in RPM

Note: *The resulting step speed is based on the 'steps' parameter in the constructor. If this does not match the number of steps for your motor, you actual speed will be off as well.*

Example:

```
Stepper1.setSpeed(10);  // Set motor 1 speed to 10 rpm
Stepper2.setSpeed(30);  // Set motor 2 speed to 30 rpm
```

onestep(*direction, stepstyle*)

Single step the motor.

Parameters:

- **direction** - the direction of rotation (**FORWARD** or **BACKWARD**)
- **stepstyle** - the style of stepping:

> Valid values for 'style' are:
>
> - **SINGLE** - One coil is energized at a time.
> - **DOUBLE** - Two coils are energized at a time for more torque.
> - **INTERLEAVE** - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
> - **MICROSTEP** - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

Example:

```
Stepper1.onestep(FORWARD, DOUBLE);  // take one step forward using double coil steppin
g
```

release()

Release the holding torque on the motor. This reduces heating and current demand, but the motor will not actively resist rotation.

Example:

```
Stepper1.release(); // stop rotation and turn off holding torque.
```