The motor shield from AdaFruit is one of the most popular Arduino shields because of what it enables you to do easily: connect multiple motors and external power sources to the Arduino without worrying about overpowers or overdraining current from the Arduino. In this article based on chapter 12 of *Arduino in Action*, author Joshua Noble discusses the features of the AdaFruit motor shield.

To save 35% on your next purchase use Promotional Code mevans1235 when you check out at www.manning.com.

You may also be interested in...

**AdaFruit Motor Shield**

The motor shield from AdaFruit is one of the most popular Arduino shields because of what it enables you to do easily: connect multiple motors and external power sources to the Arduino without worrying about overpowers or overdraining current from the Arduino. The shield is shown in figure 1.
It allows you to connect four bidirectional DC motors, two stepper motors, and also has two connections for 5 volt (V) servos. Without some serious power behind it, the shield is probably not going to work to connect all of the motors at the same time, but it does allow you to work with multiple large motors at the same time quickly and easily. Let’s look at the library that works with the shield.

**AFMotor library**

Like many shields, the motor shield comes with a library that allows you to easily control the shield and by extension any motors attached to it. We’ll first take a look at that library and show you how you can use it. To control a DC motor, you construct an instance of the `AF_DCMotor` class, passing the motor number that you’re attaching the motor to and the motor frequency:

```
AF_DCMotor motor(2);
```

How do you know the frequency of the motor that you have? Check the datasheet. Enough small DC motors are 34.8 kilohertz (kHz) that the makers of the shield made that the default. However, there are plenty of motors that are not, so it’s best to check. Once you’ve created the motor, you can start it, passing the speed in a range of 0 to 255 and then calling `run()` with either `FORWARD` or `BACKWARD` to the `run()` method.

```
motor.setSpeed(100);
motor.run(FORWARD);
```

To stop the DC motor, call `run()` and pass `RELEASE` to the `run()` method.

```
motor.run(RELEASE);
```

The stepper library works a little differently. You first create the `AF_Stepper` instance by passing the steps in a complete circle and the connection number that the stepper is connected on. The number of degrees in a step usually can be found in the datasheet. It may be listed as the number of degrees that the stepper makes per step, so you can just divide 360 by the number of steps:

```
const int STEPS = 360/1.8;
AF_Stepper stepperMotor(STEPS, 1);
```

Now, to advance the motor, call `step()` with the number of steps and the direction that you want to turn the stepper:

```
stepperMotor.step(1, FORWARD);
```

You can also set the type of step that you want the stepper motor to turn. The available options are `SINGLE`, `DOUBLE`, `INTERLEAVE`, or `MICROSTEP`.

- **SINGLE** is single-coil activation, **DOUBLE** is two coils activated at once for slightly higher torque, and **INTERLEAVE** alternates between the single and the double to get twice the resolution but half the speed, **MICROSTEP** attempts to turn the stepper smoothly to its next position using a pulse-width modulation (PWM) signal. Not all steppers can perform all of these steps, so you’ll want to check the datasheet and perhaps the AdaFruit forums for info on your particular motor.

**Working with the Motor Shield**

For this example you will need:

- 1 Arduino
- 1 AdaFruit motor shield
- 1 5 V stepper motor
- 2 AA batteries
- 2 buttons

This application uses the two buttons to control the direction of the stepper. Each button acts as a toggle, turning the stepper on and off and determining which direction it will turn, if at all. The connections are shown here. Note the battery pack attached to the motor shield to power the stepper motor. The two buttons have pull-down resistors on them to ensure that they only read HIGH when pressed.
You will also need to have the AFMotor library installed, which you can download from github at http://www.github.com/adafruit/Adafruit-Motor-Shield-library.

Listing 1 MotorDriving.pde

```c
#include <AFMotor.h>

AF_Stepper stepper(48, 1);  //from the datasheet

const int STEPPER_BWD_BUTTON = 12;
const int STEPPER_FWD_BUTTON = 11;

const int BACKWARDS = 2;
const int FORWARDS = 1;
const int STOPPED = 0;

int motorState;

void setup() {
    // nothing needed here
}

void loop() {
    int fwd = digitalRead(STEPPER_FWD_BUTTON);
    int bwd = digitalRead(STEPPER_BWD_BUTTON);
```

Figure 2 The connections for listing 1.
if(fwd == HIGH) { #A
    if( motorState == FORWARDS ) {
        motorState = STOPPED;
    } else {
        motorState = FORWARDS;
    }
}

if(bwd == HIGH) {
    if( motorState == BACKWARDS ) {
        motorState = STOPPED;
    } else {
        motorState = BACKWARDS;
    }
}

if(motorState == FORWARDS) {
    stepper.step(1, FORWARD, INTERLEAVE);
} else if(motorState == BACKWARDS) {
    stepper.step(1, BACKWARD, INTERLEAVE);
}

### #A Toggling directions

Controlling a motor is just as simple: create an instance of the AF_DCMotor class and set the controller block it's attached to in its constructor. For instance, a DC motor connected to the second motor block (marked with M2) would be declared like so:

```cpp
AF_DCMotor motor(2);
```

For the next project, we'll make a motor that has its speed controlled by a potentiometer and its direction by a pair of buttons. For this project you will need:

- 1 Arduino
- 1 AdaFruit motor shield
- 1 small DC motor
- 2 buttons
- 1 hobby servo
- 2 AA batteries
- 1 potentiometer

Sometimes you can get away with powering a stepper motor from the Arduino itself, but, with a DC motor, you'll definitely need a separate power source as shown here.
The code for this example requires the AFMotor library and the Servo library, so make sure that you have them both installed on your computer.

**Listing 2 PotToMotors.pde**

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

AF_DCMotor motor(1);
Servo servo(9);  // A

const int MOTOR_FWD_BUTTON = 12;
const int MOTOR_BWD_BUTTON = 11;
const int POTENTIOMETER_PIN = 8;

int motorSpeed;
int motorState;

const int BACKWARDS = 2;
const int FORWARDS = 1;
const int STOPPED = 0;

void setup() {
    servo.attach(9);
    // nothing needed here

    pinMode(MOTOR_FWD_BUTTON, INPUT);
    pinMode(MOTOR_BWD_BUTTON, INPUT);
    pinMode(POTENTIOMETER_PIN, INPUT);
}
```

Figure 3 The connections for listing 2

For Source Code, Sample Chapters, the Author Forum and other resources, go to www.manning.com/mevans/
```c
void loop() {
    int tmp = analogRead(POTENTIOMETER_PIN);
    int fwd = digitalRead(STEPPER_FWD_BUTTON);
    int bwd = digitalRead(STEPPER_BWD_BUTTON);

    // direction
    if(fwd == HIGH) {
        if( motorState == BACKWARDS ) {
            motorState = FORWARDS;
            motor.run( FORWARDS );
        }
    }
    if(bwd == HIGH) {
        if( motorState == FORWARDS ) {
            motorState = BACKWARDS;
            motor.run( BACKWARDS );
        }
    }

    // speed
    if( tmp != motorSpeed ) { #B
        motorSpeed = tmp;
        motor.setSpeed( motorSpeed );
    }

    // speedometer
    if( motorState == FORWARDS ) {
        servo.write( motorSpeed / 1024 * 90 + 90 );
    } else {
        servo.write( 90 - (motorSpeed / 1024 * 90) );
    }
}

#A Servo connection 1 is Arduino D9
#B Only set speed if it's changed
```

The code listing is a bit lengthy but what it does is quite simple: controlling motor speed and direction using simple components and giving some visual feedback using a servomotor. Now you have a project with a controllable motor and a speedometer as well.

**Getting a MotorShield**

There are two options for getting a MotorShield: you can buy it assembled or assemble it yourself. Buying it assembled saves you some time and costs you some money, while buying it unassembled saves you some money and gives you a chance to hone your soldering skills. If you go the latter route, then the kit will arrive with all the pieces in a bag and you’ll have to take a look at the directions for putting it together on AdaFruit site. To avoid needless repetition, I’m not going to list them here, but they’re fairly self-explanatory. The biggest things to watch out for are that you get the right capacitor in the right place and that you have the polarity of the capacitors oriented correctly. I recommend putting it together yourself, but if you want to be absolutely sure that the shield will be assembled correctly, you can buy it already assembled.

**Summary**

We’ve explored one of the better-known shields available, the motor shield from AdaFruit. It allows you to easily control up to two servo motors, four DC motors, or two stepper motors. This shield provides several very nice features in a convenient form factor, among them: pull down resistors keep motors disabled during power-up, terminal block connectors to easily hook up connections or and power, internal kickback protection diodes that allow you to use larger voltages for larger motors. That’s just the start. The advantage of using this shield is that it

For Source Code, Sample Chapters, the Author Forum and other resources, go to [www.manning.com/mevans/](http://www.manning.com/mevans/)
puts all of the functionality to work with several different kinds of motors into a single board that doesn’t need to be set up again each time your configuration changes.

Shields also often come with libraries and examples attached to them that are great ways to learn how to work with the components that the shield contains. For instance, the AdaFruit shield can be used with the AFmotor library, which is written for the Arduino and contains a wealth of knowledge of how to control the different kinds of motors the shield provides functionality for, as well as numerous examples of controlling motors.

Shields can be bought assembled or they can be assembled on your own. Often self-assembly is far cheaper than having them assembled for you, but they also introduce the possibility of making mistakes. It is, however, far more fun and educational to put things together than to have them put together for you.
Here are some other Manning titles you might be interested in:

- **Open-Source ESBs in Action**
  Tijjs Rademakers and Jos Dirksen

- **Open Source SOA**
  Jeff Davis

- **Mule in Action**
  David Dossot and John D’Emic

Last updated: January 12, 2012