Extracted from:

Programming Your Home

Automate with Arduino, Android, and Your Computer

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

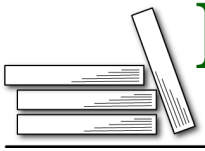
The Pragmatic Programmers

Programming Your Home

Automate with Arduino, Android, and Your Computer

Mike Riley

Edited by Jacquelyn Carter



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jackie Carter (editor)
Potomac Indexing, LLC (indexer)
Molly McBeath (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-93435-690-6
Printed on acid-free paper.
Book version: P1.0—February 2012

4.3 Dog Assembly

Take a look at the schematic in Figure 8, *Wiring diagram for the Electric Guard Dog*, on page 6. The graphic shows wiring plugging into the wave shield. The wave shield is stacked on top of the Arduino board. Note that the wave shield uses several of the pins for its own use to interact with the Arduino, which is why not all passthrough pins are available for the sketch. Closely follow the wiring diagram and you should not have a problem.

Attach the positive lead of the PIR to the 3.3v pin on the wave shield. Connect the negative lead to one of the wave shield's available ground pins. Then attach the control wire (the middle pin/wire on the PIR) to the wave shield's digital pin 12.

Next, attach the servo's positive wire to the wave shield's 5v pin. Connect the negative lead to the wave shield's other available ground pin. Finally, connect the control wire to the wave shield's digital pin 11.

For brief testing purposes, you can attach male pins to the wires and plug them directly into the sockets on the wave shield. More reliable connections can be achieved by using either male or female header pins instead. These can be obtained directly from various Arduino board suppliers. If you plan on using the wave shield exclusively for this project, you can solder the wiring permanently to the shield for the most stable electrical connection possible.

There is one more step we should take before writing the sketch. We need to either record and digitize a dog growling and barking in various ways or legally download audio samples from the Internet of snarling, barking dog sounds.

The first option takes more time and requires access to a big dog that can bark, snarl, and growl on command—with a microphone near its toothy yapper, no less! While this requires a bit more extra work, the results produce a more consistent and realistic effect. And because you know the source, playback generates a more meaningful audio cue.

The second option of searching on the Internet for a variety of angry dog audio samples is more convenient but rarely produces a consistent and believable overall effect. This is especially true when the samples are acquired from a variety of dog breeds. How can a dog have the toothy snarl of a Doberman one minute and the yapping of a miniature poodle the next? Also, downloading audio samples from the Internet has copyright implications that have to be

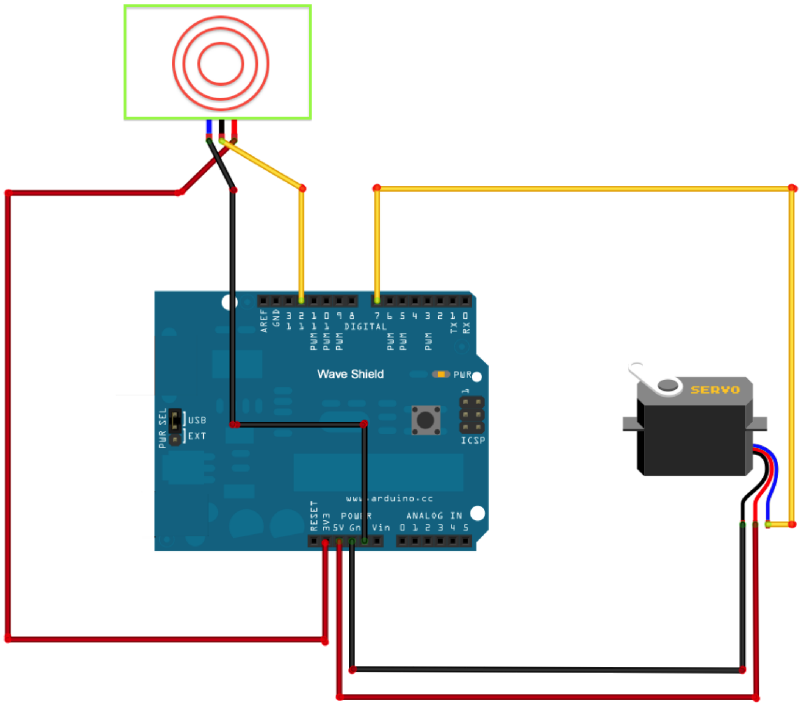


Figure 8—Wiring diagram for the Electric Guard Dog

Joe asks:

How Does a PIR Sensor Work?

A PIR detects motion by comparing two samples of infrared radiation being emitted by a body warmer than the background environment it is moving against. When either side of the sensor detects a greater value than the other, it sends a signal to the digital out pin that motion has been detected. The IR sensor at the heart of a PIR is typically covered by a dome-shaped lens that helps to condense and focus light so that it is much easier for the sensor to detect infrared variations, and thus, motion.

For a more detailed explanation of the theory behind PIRs, visit Ladyada's informative web page on the subject.[a]

a. http://www.ladyada.net/learn/sensors/pir.html

respected. One website that I recommend visiting is the Freesound Project,[7] which features a number of samples available under the Creative Commons

Sampling Plus license.

After you have obtained five audio clips using either approach, you need to convert them to a format the wave shield can interpret. Based on the conversion instructions on Ladyada's website,[8] samples must not exceed a 22KHz 16-bit mono PCM (WAV) format. You want the highest audio quality possible, and there should be plenty of space on the SD card to store them. The audio clips you select for the project should not exceed five seconds in duration so they appear more synchronized with the servo motion when the audio is played back.

You can use an audio editor like Audacity to import and convert and save your audio clips to the correct format.[9] Make sure they are compatible by copying the converted files to the wave shield's SD card and running the `dap_hc.pde` sketch posted on Ladyada's website.[10] Note that we're going to make one change to Ladyada's wave shield demo sketch. Instead of the newer `wavehc` library it uses, we are going to use the older `AF_Wave` library. That way, we can use Arduino community forum member avandalen's MediaPlayer library[11]—it makes working with wave shield sound files far easier. We will take a closer look at this library and another Arduino community contributor's library for servos when we write the sketch in the next section.

4.4 Dog Training

The sketch we write will monitor the PIR for any motion events. If movement is detected, the shield will randomly play one of five different audio files stored on the wave shield's SD card. Simultaneously, the servo motor rotates up to 150 degrees, depending on the sound effect being played back. Attach a wooden rod to the servo gear and the servo's rotation will move the rod up and down. When the rod is positioned behind a curtain, it will give the illusion of a dog's snout attempting to nudge the curtain aside so it can see who's at the door or window.

To begin, we need to include the `MediaPlayer.h` header file along with its two dependencies, `pgmspace.h` (part of a memory management library included in the Arduino's standard installation) and `util.h` (part of the original wave shield's `AF_Wave` library). Because the MediaPlayer class relies on the `AF_Wave` library,

7. http://www.freesound.org
8. http://www.ladyada.net/make/waveshield/convert.html
9. http://audacity.sourceforge.net/
10. http://www.ladyada.net/make/waveshield/libraryhc.html
11. http://www.arduino.cc/playground/Main/Mediaplayer

make sure you have already downloaded, unzipped, and copied the uncompressed AF_Wave folder into the Arduino's libraries folder.[12]

Next, create a new sketch in the Arduino IDE called ElectricGuardDog. Download the MediaPlayer library from the Arduino playground website;[13] extract the zip archive; and place the unzipped MediaPlayer.h, MediaPlayer.pde, and MediaPlayerTestFunctions.pde files into the ElectricGuardDog folder created by the Arduino IDE when it created the ElectricGuardDog.pde file. If you downloaded the project files for the book, the Mediaplayer library file dependencies have already been prebundled for you. The Mediaplayer library allows us to control audio file playback very easily.

We will also need to call upon another custom library to operate the servo motor. If you try to compile the sketch using the standard Arduino Servo class, the program will fail with this error:

```
Servo/Servo.cpp.o: In function `__vector_11':
/Applications/Arduino.app/Contents/Resources/Java/libraries/Servo/Servo.cpp:103:
multiple definition of `__vector_11'

AF_Wave/wave.cpp.o:/Applications/Arduino.app/
Contents/Resources/Java/libraries/AF_Wave/wave.cpp:33: first defined here
```

What's going on here? The AF_Wave library is taking over the vector interrupt as the standard Servo library. Fortunately for us, Arduino community contributor Michael Margolis has written a library that gives the Arduino the ability to control up to eight servo motors simultaneously. By doing so, his library also circumvents the duplicate resource problem exhibited by the original Servo library when combined with a wave shield.

Download the ServoTimer2 library,[14] unzip it, and copy the ServoTimer2 folder into the Arduino libraries folder. Keep in mind that each time you add a new library to the Arduino libraries folder, you need to restart the Arduino IDE so the Arduino's avr-gcc compiler will recognize it.

After the wave shield's AF_Wave and servo motor's ServoTimer2 library dependencies have been satisfied, add these references to the beginning of the sketch:

Download ElectricGuardDog/ElectricGuardDog.pde

```
#include <avr/pgmspace.h>
#include "util.h"
#include "MediaPlayer.h"
#include <ServoTimer2.h>
```

12. http://www.ladyada.net/media/wavshield/AFWave_18-02-09.zip
13. http://www.arduino.cc/playground/Main/Mediaplayer
14. http://www.arduino.cc/playground/uploads/Main/ServoTimer2.zip

Create several variables to store Arduino pin assignments and sensor/actuator starting values.

Download ElectricGuardDog/ElectricGuardDog.pde

```
int ledPin        = 13;  // on board LED
int inputPin      = 12;  // input pin for the PIR sensor
int pirStatus     = LOW; // set to LOW (no motion detected)
int pirValue      = 0;   // variable for reading inputPin status
int servoposition = 0;   // starting position of the servo
```

Next, create two objects constructed from the MediaPlayer and ServoTimer2 libraries to more easily manipulate the servo motor and audio playback.

Download ElectricGuardDog/ElectricGuardDog.pde

```
ServoTimer2 theservo;    // create servo object from the ServoTimer2 class
MediaPlayer mediaPlayer; // create mediaplayer object
                         // from the MediaPlayer class
```

Assign the variables we created to the Arduino pinModes in the sketch's setup() routine. Establish a connection to the Arduino IDE serial window to help monitor the motion detection and audio playback events. Call the Arduino's randomSeed() function to seed the Arduino's random number generator. By polling the value of the Arduino's analog pin 0, we can generate a better pseudorandom number based on the electrical noise on that pin.

Download ElectricGuardDog/ElectricGuardDog.pde

```
void setup() {
  pinMode(ledPin, OUTPUT);   // set pinMode of the onboard LED to OUTPUT
  pinMode(inputPin, INPUT);  // set PIR inputPin and listen to it as INPUT
  theservo.attach(7);        // attach servo motor digital output to pin 7
  randomSeed(analogRead(0)); // seed the Arduino random number generator
   Serial.begin(9600);
}
```

With the library, variable, object, and setup initialization out of the way, we can now write the main loop of the sketch. Essentially, we need to poll the PIR every second for any state changes. If the PIR detects motion, it will send a HIGH signal on pin 12. When this condition is met, we power the onboard LED and send a motion detection message to the Arduino IDE's serial window.

Next, we generate a random number between 1 and 5 based on the seed we created earlier. Based on the value generated, we then play back the designated audio event and move the servo motor a predefined amount of rotation. After that, we wait a second before returning the servo to its starting position and run the loop again. If the PIR fails to detect motion (that is, if the signal on pin 12 is LOW), we turn off the onboard LED, send a No motion message to the serial window, stop the audio playback, and set the pirStatus flag to LOW.

Download ElectricGuardDog/ElectricGuardDog.pde

```
void loop(){
  pirValue = digitalRead(inputPin); // poll the value of the PIR
  if (pirValue == HIGH) {           // If motion is detected
    digitalWrite(ledPin, HIGH);         // turn the onboard LED on
    if (pirStatus == LOW) {                    // Trigger motion
      Serial.println("Motion detected");

      // Generate a random number between 1 and 5 to match file names
          // and play back the file and move the servo varying degrees
      switch (random(1,6)) {
        case 1:
          Serial.println("Playing back 1.WAV");
          theservo.write(1250);
          mediaPlayer.play("1.WAV");
          break;
        case 2:
          Serial.println("Playing back 2.WAV");
          theservo.write(1400);
          mediaPlayer.play("2.WAV");
          break;
        case 3:
          Serial.println("Playing back 3.WAV");
          theservo.write(1600);
          mediaPlayer.play("3.WAV");
          break;
        case 4:
          Serial.println("Playing back 4.WAV");
          theservo.write(1850);
          mediaPlayer.play("4.WAV");
          break;
        case 5:
          Serial.println("Playing back 5.WAV");
          theservo.write(2100);
          mediaPlayer.play("5.WAV");
          break;
      }

      delay(1000);          // wait a second
      theservo.write(1000); // return the servo to the start position
      pirStatus = HIGH;     // set the pirStatus flag to HIGH to stop
                            // repeating motion
    }
  } else {
    digitalWrite(ledPin, LOW); // turn the onboard LED off
    if (pirStatus == HIGH){
      Serial.println("No motion");
      mediaPlayer.stop();
      pirStatus = LOW;      // set the pirStatus flag to LOW to
                            // prepare it for a motion event
```

```
    }
  }
}
```

Save the code as ElectricGuardDog.pde and open up the newly created ElectricGuardDog folder containing the ElectricGuardDog.pde source file. Place the unzipped MediaPlayer files into the ElectricGuardDog directory. Double-check that the uncompressed ServoTimer2 library files are in the Arduino libraries directory.

Reopen the Arduino IDE, load up the ElectricGuardDog.pde file, and click the Verify icon in the Arduino IDE toolbar. If everything compiled without errors, you have entered the code correctly and placed the dependent library files in the correct locations. If not, review the error messages to see what dependencies may be missing and correct accordingly.

With the sketch compiled successfully, we're ready to test and tweak the code.