```
// modified from:
// midi_in_state.ino
// version 2015-06-01
// Arduino MIDI tutorial
// by Staffan Melin
// http://libremusicproduction.com/
// libraries

/* We will use the SoftwareSerial library instead of the Serial library, as this will let us control which pins our MIDI interface is connected to*/
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3);  // RX, TX

/* Use #define to give names to constant values.  The compiler will replace references to these constants with the defined value*/
#define PIN_LED 13
#define PIN_RELAY 7
#define PIN_RELAY_B 8

/*Use #define to give names to MIDI command values.  Value 144 is the value for the midi "NOTE ON" command*/
#define MIDI_NOTE_ON 144
#define MIDI_NOTE_OFF 128

// Filter MIDI events on channel and note number
const int filterChannel = 1; // MIDI channel 2.  Channel 1 starts at value 0 up to 15 (Channels 1-16)
const int filterNote = 24; /*Used to indicate audio track armed (blinks) and then recording (solid). Using this seems to exclude light during midi recording, which responds to 25.  Will omit this requirement for our purposes.*/

// Use #define to give names to values 0-2.  We use these as states for a switch statement.
#define STATE_NONE 0
#define STATE_COMMAND_SENT 1
#define STATE_NOTE 2
int state;

/*Received MIDI data is sent serially in bytes (8 bits).  A byte stores an 8-bit binary number.  Binary uses only a 0 or 1 for each of the 8 digits (bits).*/
byte midiByte;
byte midiChannel;
byte midiCommand;
byte midiNote;
byte midiVelocity;

void setup() {

  mySerial.begin(31250);
  delay(30);
```

```
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_RELAY, OUTPUT);
  pinMode(PIN_RELAY_B, OUTPUT);
  digitalWrite(PIN_LED, LOW);
  digitalWrite(PIN_RELAY, LOW);
  digitalWrite(PIN_RELAY_B, LOW);

  state = STATE_NONE;
}

void loop () {

  // Is there any MIDI waiting to be read?

  if (mySerial.available() > 0)
  {
    // Read MIDI byte

    midiByte = mySerial.read();

    switch (state)
    {
    case STATE_NONE:

    /*Extract the channel info.  First byte contains both midiCommand and midiChannel information.
    The first 4 digits of the byte contain the command (i.e. NOTE_ON value 144 = 10010000 in
    binary).  The last 4 digits of the byte contain the channel (i.e. Channel 1 = 00000001 in binary).
    Bitwise '&' operation pulls out pertinent command and channel number by multiplying each of
    the bits together*/
      midiChannel = midiByte & B00001111;
      midiCommand = midiByte & B11110000;

      if (midiChannel == filterChannel)
      {
        if (midiCommand == MIDI_NOTE_ON)
        {
          state = STATE_COMMAND_SENT;
        }
      }
      break;

    case STATE_COMMAND_SENT:
      midiNote = midiByte;  //Second byte sent by midi is the pitch - we don't use it in this application
      state = STATE_NOTE;
      break;

    case STATE_NOTE:
```

```
        midiVelocity = midiByte;  /*Third byte sent by midi is the velocity.  In the case of Logic pro
        recording light control surface, 127 is sent when recording starts and 0 is sent when it stops*/
        state = STATE_NONE;

        if (midiCommand == MIDI_NOTE_ON && midiVelocity == 127) //Recording started
        {
          digitalWrite(PIN_LED, HIGH);
          digitalWrite(PIN_RELAY, HIGH);
          //Next, turn second relay on for only a short time to simulate a momentary switch
          digitalWrite(PIN_RELAY_B, HIGH);
          delay(300);
          digitalWrite(PIN_RELAY_B, LOW);
        }

        if (midiCommand == MIDI_NOTE_ON && midiVelocity == 0)  //Recording stopped
        {
          digitalWrite(PIN_LED, LOW);
          digitalWrite(PIN_RELAY, LOW);
          //Next, turn second relay on for only a short time to simulate a momentary switch
          digitalWrite(PIN_RELAY_B, HIGH);
          delay(300);
          digitalWrite(PIN_RELAY_B, LOW);
        }
        break;

      } // switch

    } // mySerial.available()

} // loop
```