PowerBeats
Personal Noise Dosimeter
Instruction Manual

TABLE OF CONTENTS

Introduction

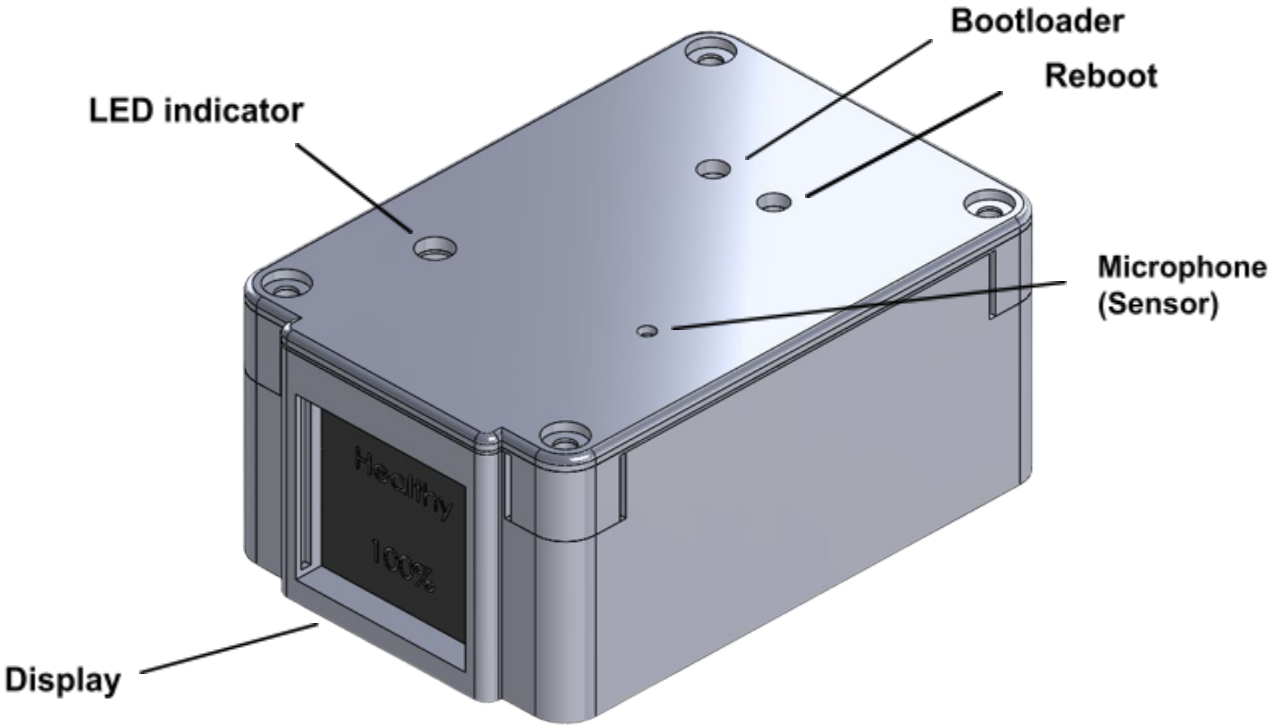
What is a noise dosimeter?

A noise dosimeter is a device that detects, processes, and determines outputs of sound data for the purpose of informing users of noise exposure over time in order to protect them from hearing loss. Our device is a noise dosimeter that follows OSHA regulations and indicates the duration of exposure based on the intensity level throughout the day. We monitor the level of exposure to damaging noises through an allowance system. The allowance is shown as a percent bar that decreases throughout the day if the device senses a sound pressure above 80 dBA. This dosimeter uses a microphone to detect the sound waves. In addition, the circuit includes a series of analog filters to mimic the human auditory range of 20Hz - 20kHz, as also required by OSHA. By having a better understanding of the noise people are exposed to on a daily basis, they can take the necessary steps to protect their hearing

Getting Started

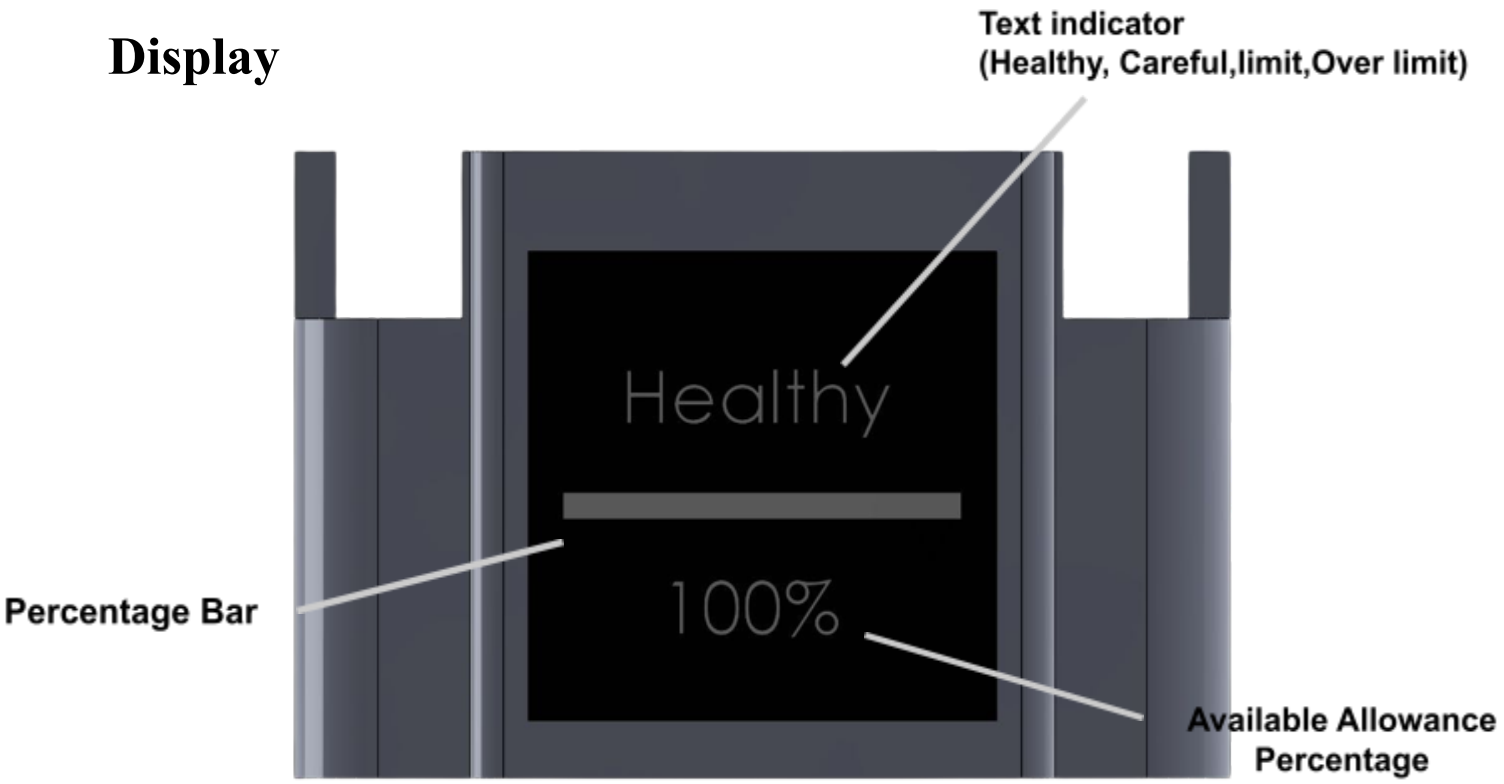
NOISE DOSIMETER OVERVIEW

Exterior

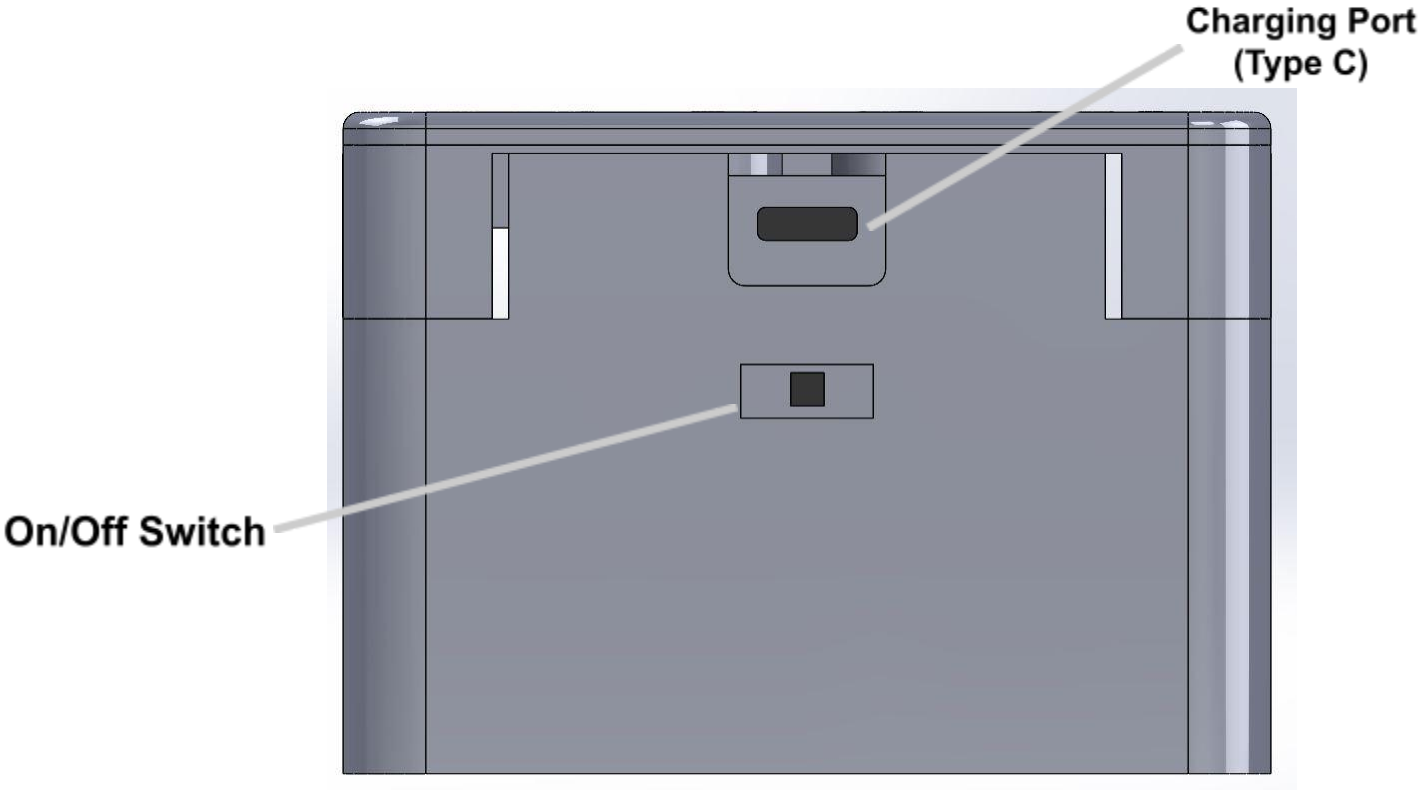


u

Display



Charging



How To Use

Basic Operation

1.) **Turning on the device:** To turn on the device, use a narrow flat pin to move the switch from the left to the right.

2.) **Display:** When the device is first turned on, the screen will display the text "Healthy" and a full allowance bar with the percentage at 100%. The allowance bar will start to decrease when registering a sound pressure above 80 dBA.

3.) **LED indicator**: When in a noisy environment of above 80 dBA the LED will turn on and remain on until registering a reading less than the 80 dBA threshold. If the noise environment is **ABOVE 100 dBA**, the LED will start flashing indicating that you are in a severely dangerous location. It is crucial to evaluate the premises immediately to avoid hearing damage. These sound pressure levels are harmful to the ear and will deplete your allowance quickly.

4.) **Resetting the Allowance:** The device is intended to be left on for the entire day once the switch is moved to the right. At the end of the day, you can reset the allowance by simply turning the device off and then on again. The switch was designed to require a narrow flat pin or other means to prevent accidental allowance reset before the end of the day.

5.) **Battery life and Charging:** The device has an approximate battery life of 28 hours. After the day is complete, you can charge the device using any standard Type C charger. It is possible to use the device while the battery is charging.

Other features

Sound Data: For those who want to perform further analysis of their sound data, the device is equipped with an 8GB microSD card to record data above the 80 dBA threshold. To access the memory card, use a Phillips head screwdriver to remove the four screws at the top of the case. **The Sound_User.txt** file will be saved in a folder named "**Noise_Dosimeter**." The file will contain all the timestamps when the device registered sound above 80 dBA, the actual dBA reading, and the allowance percentage at that point in time.

Troubleshooting

What to do if:

1.) **Screen doesn't turn on?**
   - **a.)** Check to see if the switch is properly positioned to the right.
   - **b.)** Check to see if the device is out of charge by plugging the device using a Type C cable. The device should be on and functioning.
   - **c.)** Perform a software reboot. (Appendix A) by copying and pasting code into an Arduino IDE script after downloading the proper libraries.
     - **i.)** Wire.h
     - **ii.)** Adafruit_SSD1306.h
     - **iii.)** SD.h
     - **iv.)** iostream
     - **v.)** algorithm
     - **vi.)** elapsedMillis.h
     - **vii.)** TimeLib.h
     - **viii.)** ESP32 Library by espressif https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

2.) **Screen is at 100% but is not going down?**
   - **a.)** If the user is not in a loud environment, the screen will not update and will stay at 100%. A good indicator that you are in a potentially hearing damaging environment is when the LED turned on since the device only deducts from your allowance if the device registered sound pressure above 80 dBA
   - **b.)** Perform a software reboot (Appendix A)

3.) **For Seeed XIAO ESP32C3 Support**
   - **a.)** Refer to seeed studio website for help connecting, resetting, and rebooting device. https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/

If the above-mentioned steps still did not resolve your issue, then there is likely a wiring issue with your device, and it should be inspected by an electronics specialist who can repair it.

Technical Support Contact

For any additional questions or resources related to the device please feel free to contact us at powerbeats.ccny@gmail.com

APPENDIX A

For more information on the OSHA standard used in developing program and the noise dosimeter, please referred to the following link: https://www.osha.gov/noise/standards

---

How to perform a software reboot.

- Download the latest version of the Arduino IDE and all the required library found in **Final_DoseClip_Noise_Dosimeter.ino**

- Upload the code to the Arduino IDE and use a flat pin to press down on the two similarly shaped holes before proceeding with the code. At the other end of the holes, there are two buttons used for configuring the device.

CODE

Final_DoseClip_Noise_Dosimeter.ino

```
using namespace std;
// Require Library for code to run. Download first before uploading
code into microcontroller
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <SD.h>
#include <iostream>
#include <algorithm>
#include <elapsedMillis.h>
#include <TimeLib.h>
```

```
// Objects //
elapsedMillis timeElapsed;
File myFile;
#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels
#define BAUD_RATE 115200
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
const int micPin    = A3;    // Collects the sound data and process
sound
const int interval = 1000; // 1-sec Slow interval
float Allowance_Start = 100;
// used in RMS average.
int   Count=0;
float dBA=0;
float sum_square=0;
/* Definition */
// initialization of device
void initialization(int ADRR, int CS_pin){
  // Configuration of the display bar
 int width = 128;
 const int Color = 0xFFFF; // White in RGB565 format
 const int x = 0;
 const int y = 32;
 const int height = 10; // Height of Bar
 Serial.print("Initializing SD card...");
  if (!SD.begin(CS_pin))
   {
     Serial.println("initialization failed!");
     while (1); }

 Serial.println("initialization Complete."); Serial.println("");
Serial.println("");
 Serial.print("Initializing OLED Display...");
   if(!display.begin(SSD1306_SWITCHCAPVCC, ADRR))
     {
```

```
      Serial.println(F("SSD1306 allocation failed"));
      for(;;);
    }
 Serial.println ("Initializing Complete.");
Serial.println("");Serial.println("");
  /* Data logging Section */
 SD.mkdir("/Noise_Dosimeter"); // Creates a folder to place the user
information
 if(SD.exists("/Noise_Dosimeter/Sound_User.txt"))
   {
    myFile = SD.open("/Noise_Dosimeter/Sound_User.txt",
FILE_APPEND); // Appends data into file
      {
        myFile.printf("\n");
        myFile.printf("Allowance has been reset\n");
        myFile.printf("%-12s%-10s%-19s\n", "Time", "dBA",
"Allowance");
        myFile.printf("%-12s%-10s%-19s\n", "----", "---",
"---------");
        myFile.close();

        Serial.println("File is ready to append.");
        Serial.println("");
      }
   }
 else
    {
      myFile = SD.open("/Noise_Dosimeter/Sound_User.txt",
FILE_WRITE); // Creates an txt file for collecting data.
      if (myFile)
        {

          myFile.printf("%-12s%-10s%-15s\n", "Time", "dBA",
"Allowance");
```

```
            myFile.printf("%-12s%-10s%-15s\n", "----", "---",
"---------");
            myFile.close();

            Serial.println("File has been created.");
            Serial.println("");
          }
      }
```

```
 display.clearDisplay();
 display.drawRect(x,y, (width - 1), height, Color);
 display.fillRect(x,y, (width - 1) * 1, height, Color); // width *
fraction of the allowance
 display.setTextSize(2);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(5, 6);
 display.println("Healthy");
 display.setTextSize(2);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(30, 50); // x,y
 display.print(100.00);
 display.println("%");
 display.display();
}
// For the conversion of A/D to volts to dBA
float bits_2_Vsquare(float Bits){
 // float Bits is an input from analogRead()
 // bias_offset is from half the battery capacity.
 // Vcc = 3.3V so bias_offset is 1.65;
 float dev_resolution = 3.3/4096;
 float bias_offset    = 1.65;
 float Bitsv  = (Bits * dev_resolution) - bias_offset;
 float square = pow(Bitsv,2);
 return square;
```

```
}
float RMS_v(int &Count, float &sum_square){
 // Conversation AD reading into Root Mean Square (RMS)
 float RMS_v = sqrt(sum_square/Count);
 Serial.print("Summed volts: "); Serial.println(sum_square);
 Serial.print("Samples: ");      Serial.println(Count);
 Serial.print("RMS: ");          Serial.println(RMS_v);
 return RMS_v;
}
float RMSv_to_dBA(float RMSv){
     float x = 20*log10(RMSv/0.00002);
     float x_square = pow(x,2);
   if (RMSv > 0.316274 && RMSv < 0.354264) // 80 dBA - 90 dBA
       {
         dBA = (-3.1672*x_square)+(545.23*x)-23372 ;
Serial.print("dBA: "); Serial.println(dBA);
       }
   else if (RMSv >= 0.354264) // 90 dBA and above
       {
         dBA =  (2.1571*x)-93.162; Serial.print("dBA: ");
Serial.println(dBA);
       }
   else
       {
         dBA = 0; Serial.println("dBA was less than 80. Allowance not
taken off"); // Anything below 80 is not necessary and is assigned a
dummy variable.
         Serial.println("");
       }
   return dBA;
}
}
// For configuration of the screen and allowance
float Allowance(float dBA, float Allowance_Start) {
 float factor = 1.6667; // 1-sec measurements ;
```

```
 float Weighting =
factor*(1/(exp(-0.13862944*dBA)*exp(18.65043535)));  // 1-sec
measurements
 float Allowance = Allowance_Start - Weighting;

 Serial.print("Weighting: ");  Serial.print(Weighting,5);
Serial.print("; ");
 Serial.print("Allowance: ");  Serial.println(Allowance);
 Serial.println("");
 return Allowance;
}
void Allowance_Disp_Setting(float &allowance){
 int width = 128;
 const int Color = 0xFFFF; // White in RGB565 format
 const int x = 0;
 const int y = 32;
 const int height = 10; // Height of Bar
 float allowance_percent = (allowance/100);
 display.clearDisplay();
 display.drawRect(x,y, (width - 1), height, Color);
 display.fillRect(x,y, (width - 1) * allowance_percent, height,
Color); // width * fraction of the allowance
 display.setTextSize(2);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(5, 6);
 //  Display text when a certain percentage has been depleted
 if (allowance > 60)
   {
     display.println("Healthy");
   }
 else if (allowance > 2)
   {
     display.println("Careful!");
   }
 else if (allowance > 0)
```

```
   {
     display.println("Limit");
   }
 else
   {
     display.println("Over Limit");
   }
  display.setTextSize(2);
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(35, 50); // x,y
 display.print(allowance,2);
 display.println("%");
 display.display();
}
// For Data logging
void Data_Log(long int time,String Filename,float dBA, float Allow){
/* Time      = elapse time after device is one
   Filename  = Naming the file to append dBA and Allowance
percentage.
 */
 Serial.print("Time:");
 // To convert elapse time into hour, minutes and seconds
 unsigned long hours =   (time /3600000UL) % 24;
 unsigned long minutes = (time /60000UL) % 60;
 unsigned long seconds = (time /1000UL) % 60;
 Serial.print(hours);   Serial.print(":");
 Serial.print(minutes);  Serial.print(":");
 Serial.println(seconds);
 myFile = SD.open(Filename, FILE_APPEND);
  if (myFile)
   {
     myFile.printf("%02lu:%02lu:%02lu    %-10.2f%-10.2f\n", hours,
minutes, seconds, dBA, Allow); //Format of the time in H:M:S
   }
}
```

```
// Main Portion of the Code
void setup()
{
 Serial.begin(BAUD_RATE);
 initialization(0x3C,4);
  pinMode(D7, OUTPUT);    // Turning on the LED
 digitalWrite(D7,LOW);  // WHen the device is on, the LED is
initially off
}
void loop(){
 long int time = millis();  // elapsed time
  float square =  bits_2_Vsquare(analogRead(micPin));
 sum_square  += square; Count++;   // used in calculating RMS
 if (timeElapsed > interval)
   {

     float RMSv =  RMS_v(Count,sum_square); // Returns RMS voltage
     float dBA  =  RMSv_to_dBA(RMSv);       // Converts from RMS
volts to dBA based on calibration
     if (dBA >= 80) { // The 80 dBA threshold is from OSHA standard
requirement

         digitalWrite(D7,HIGH); // LED is turn on
         float Allow = Allowance(dBA, Allowance_Start);  // This is
the portion of the code where the allowance gets depleted
         Allowance_Disp_Setting(Allow);  // Display is updated

         Allowance_Start = Allow;  // Updates the allowance
percentage

        if ( dBA >= 100){
         // Code to blink the LED once the dBA reading is at 100 and
above.
```

```
          digitalWrite(D7,HIGH);
          digitalWrite(D7,LOW);
          digitalWrite(D7,HIGH);
          digitalWrite(D7,LOW);

         }
         Data_Log(time,"/Noise_Dosimeter/Sound_User.txt",dBA, Allow);
// where the data gets append
       }
       else {
         // When the reading is less than 80 then the LED is turned
off.
         // LED should only be on when the reading is at 80 and above
         digitalWrite(D7,LOW);       }
       // Reset parameter for next iteration used in calculating
data.
       square=0;
       sum_square=0;
       timeElapsed = 0;
       Count=0;
   }
}
```