# Overview  The active buzzer has built-in oscillating source, so it will beep as long as it is wired up, but it can only beep with fixed frequency.

# Step 1: Connect It

Use the following wiring diagram to connect the sensor to your Raspberry Pi



**Important Notes:**

* The wire colors do not matter. Just use whatever is available at your station.
* **Make sure that you are plugging the wires into the correct location on both the Pi and the Sensor (especially the power and ground)**
* The wires move around a lot, so make sure they are firmly in place when you are testing

# Step 2: Code It

We have provided you with a Python code template that can run this sensor/device. The code is located at:

**/home/pi/Documents/rpi-iot-demos/active\_buzzer.py**

Use the Geany editor to open this file (NOTE: this will be slow, so be patient). Once it is open, take some time to read the code to see how it works.

Once you are ready, do the following:

1. Find the function called loop().
   1. This function is called whenever the program starts. It runs forever.
2. Modify this function so that it beeps for 0.5 seconds. To do this, you will make a new function called . . . wait for it . . . beep().

def beep(time\_in\_seconds):

on()

time.sleep(time\_in\_seconds)

off()

time.sleep(time\_in\_seconds)

def loop():

while active == True:

beep(0.5)

# Step 3: Run / Test It

1. To run the script, open a Linux console and navigate to the folder with your code
   1. Helpful Linux Commands:  
      **ls** lists the contents of the directory  
      **cd <folder\_name>** opens a folder (don’t type the < > characters)  
      **cd ..** exits the folder you are currently in  
      **cd ~** takes you back to your home folder
2. Run the python program by typing the following:

**python3 active\_buzzer.py**

1. Do you hear the beeping?
   1. If it doesn’t work, make sure that your wiring is correct and that you see lights on the sensor / device
   2. Can you modify this program so that it beeps an interesting pattern?
2. **To exit the program, press Ctrl-C in the terminal**

# Step 4: Make It Controllable

To make your program listen for MQTT messages, modify your main program to look like the following:

# --------------------------------------------

# Main Program Starts Here

# --------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

setup(Buzzer)

try:

comm.connect(channels=[("jbcs/buzzer", 0)])

comm.listen(on\_message)

print("Buzzer Running")

loop()

except KeyboardInterrupt:

destroy()

Now, create a new function above your main program. This function will process any messages transmitted to your program.

# --------------------------------------------

# This is the code that will be run every time

# a message is received

# --------------------------------------------

def on\_message(client, userdata, msg):

global active

print("Received:", msg)

if msg == "OFF":

active = False

elif msg == "ON":

active = True

# Step 4: Make It Control Something

To make your program send MQTT messages, modify your main program to look like the following:

# --------------------------------------------

# Main Program Starts Here

# --------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

comm.connect()

setup()

try:

print("Program Running. Waiting for a Button Press")

loop()

except KeyboardInterrupt:

destroy()

Now, go to the OnStateChange() function you modified earlier. You can use the comm.send function to send a message when the button is pressed, etc.

comm.send("CHANNEL NAME GOES HERE", "MESSAGE GOES HERE")