BEN NUTTALL
Raspberry Pi's Community Manager, creator of the GPIO Zero library, Jam master, and the Foundation's resident Python expert!
twitter.com/ben_nuttall

REMOTELY CONTROL GPIO WITH GPIO ZERO

GPIO Zero is a very powerful tool, and now you can use it when you're not even on the Raspberry Pi!

The GPIO Zero Python library not only makes programming simple electronics easier; it comes with some advanced features. These offer seamless interfacing between different devices, while helping you progress along the Python learning curve. One useful thing about GPIO Zero is that you can choose which low-level pin library to use, allowing you to take advantage of the power of another library as required, without having to rewrite your code. By default, Ben Croston's **RPi.GPIO** library is used, and that's fine for most purposes. One of the supported alternative libraries is Joan 2937's **pigpio** library, which supports remote GPIO. This allows you to remotely control the GPIO pins of a Pi over a network. You can control the pins from a PC or Mac, or from another Pi, and even use the GPIOs of multiple Pis within the same script.

This month, GPIO Zero v1.4 was released, stabilising the remote pins syntax. This guide is written for v1.4 and will not work on earlier versions. Make sure you upgrade before you start: open a Terminal and enter **sudo apt update && sudo apt install python3-gpiozero**.

A simple GPIO Zero Python script looks like this:

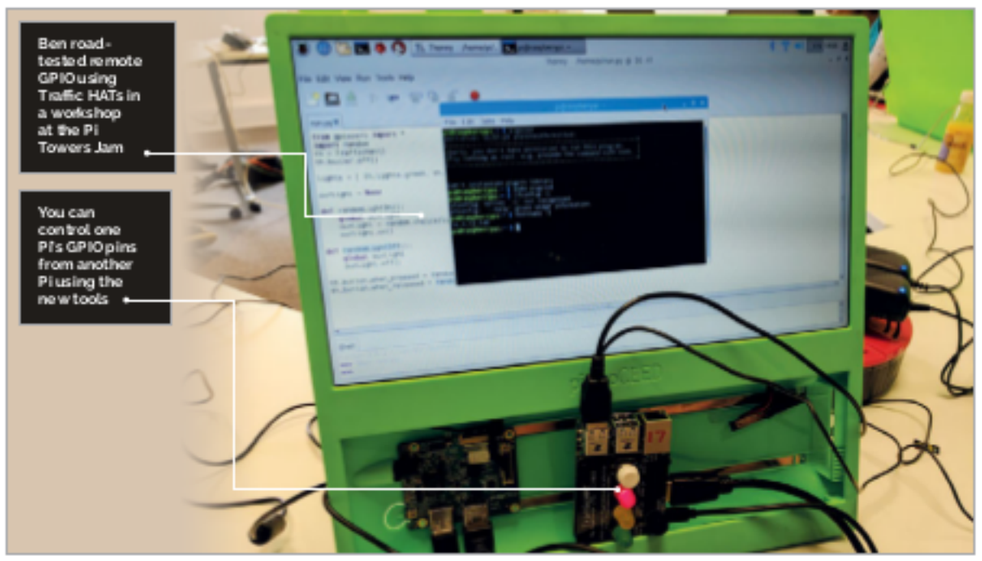
```
from gpiozero import Button, LED
from signal import pause

btn = Button(2)
led = LED(17)

led.source = btn.values

pause()
```

Running this script on a Pi will work as expected: a button connected to pin 2 (BCM numbering) will light an LED connected to pin 17 when pressed. However, when configured correctly, running this same script can control the pins of a Pi over the network.



Pin factories

The way GPIO Zero wraps around low-level pin libraries is by providing a pin factory. By default, an **RPi.GPIO**-based factory is used, and when you ask for a pin, the factory gives you a connection to it using the chosen pin library. A **pigpio** pin factory can be used on its own (simply use the **pigpio** library instead of **RPi.GPIO**), but if an IP address is provided too, this can be used to remotely control a Pi's pins.

To run the above script (unchanged) on a remote Pi, the Pi needs to be configured to accept remote connections. This can be done using the Raspberry Pi configuration tool (via GUI or **sudo raspi-config**), by enabling Remote GPIO under Interfaces. Otherwise, the Pi needs to have the **pigpio** daemon running, by entering **sudo pigpiod** in a Terminal. Finally, look up the Pi's IP address with **hostname -I**. Now return to the Pi you're running the script from, and instead of running the code normally (like **python3 led_button.py**), set two environment variables in the same command, using the remote Pi's IP address:

```
GPIOZERO_PIN_FACTORY=pigpio
PIGPIO_ADDR=192.168.1.5
python3 led_button.py
```

Now, when the script runs, the GPIO commands are executed on the remote Pi over the network.

An alternative to running a script from the command line is to set the environment variables before launching your Python editor. For example:

```
GPIOZERO_PIN_FACTORY=pigpio
PIGPIO_ADDR=192.168.1.5
idle3 &
```

You can also export these variables in your **.bashrc** file. See **magpi.cc/2qd2MEb** for more information.

Hot-swapping pin factories

The previous example showed how to set the default pin factory. Unless otherwise specified, any GPIO devices created will be connected to pins created by this default pin factory. Alternatively, you can specify a pin factory (and with **pigpio**, an IP address) within the Python code. There are two options for doing this.

You can create a pin factory instance, and pass that in as you create a new object, like so:

```
from gpiozero import LED, Button
from gpiozero.pins.pigpio import PiGPIOFactory
from signal import pause

factory = PiGPIOFactory('192.168.1.5')

btn = Button(2) # local RPi.GPIO pin
led = LED(17, pin_factory=factory) # remote pin

led.source = button.values

pause()
```

Alternatively, you can change the default pin factory in the middle of your script, like so:

```
import gpiozero
from gpiozero import LED, Button
from gpiozero.pins.pigpio import PiGPIOFactory
from signal import pause

btn = Button(2) # local RPi.GPIO pin
gpiozero.Device.pin_factory = PiGPIOFactory('192.168.1.5')
led = LED(17) # remote pin

led.source = button.values

pause()
```

Press the button on your Pi and watch the LED light up on the remote Pi. With no environment variables set, **RPi.GPIO** is used as the default pin factory. When the button is created, it uses **RPi.GPIO** to address a local pin. The default pin factory is replaced with **pigpio**, connecting to a particular IP address, and the LED is created on pin 17, which now refers to the remote Pi.

While this can be a confusing concept, it's quite simple once you get used to the idea, and it could be very useful in many projects. You can even run this code on a PC (not a Raspberry Pi) and use it to control a Pi on the network. Any platform (Windows, Mac or Linux) will work, as long as you have Python, **pip**, **GPIO Zero**, and **pigpio** installed. For full instructions, head over to **rpf.io/remotegpio**.

Security

It's worth pointing out that allowing remote GPIO connections over the network can be risky. You probably shouldn't do this in a real project on a network with other users. However, you can take precautions to make it more secure. An easy method is to only allow remote connections from a particular IP address when launching the **pigpio** daemon: **sudo pigpiod -n 192.168.1.4**.

Check out some remote GPIO recipes and more on the GPIO Zero documentation at **magpi.cc/2qd2MEb**.

LEARN MORE ABOUT GPIO ZERO

GPIO Zero is an amazing tool for creating simple electronics and making your projects just that little bit easier. Check out our Essentials book, *Simple Electronics with GPIO Zero*, to learn more: **magpi.cc/2bA3ZP7**.