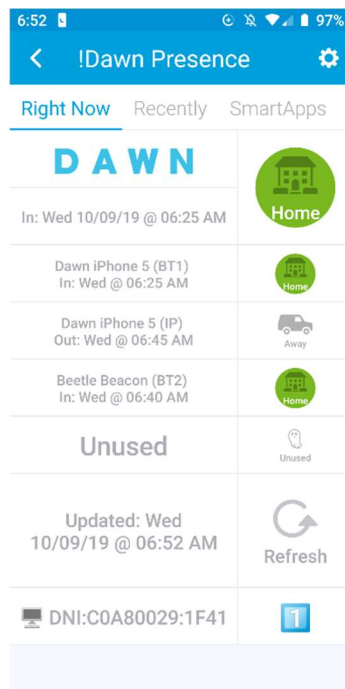


# Unified Presence Monitor V 1.0

A solution that looks at combination of WiFi, Bluetooth or BLE signals to determine if a given person is present. Presence is marked in the SmartThings device handler as shown below.



## OVERVIEW

This project consists of two pieces. A Python script with supporting shell script that nominally runs on a Raspberry Pi. This script is configured with people, devices and device addresses to monitor for presence. One person may have up to four devices. All the devices must be "out" for the person to be considered out. Any one device in and the person is considered in. The second piece is a device handler for SmartThings that receives the information via a JSON file and marks the persons presence accordingly in SmartThings.

I built this using a RPi 4B but have also tested it with a RPi Zero W. These steps are based on a fresh install (Buster) so may need to be adjusted based on your individual version.

## NETWORK PREP

You will need to assign a static IP address to the RPi for this to work reliably. You can either configure a static IP or use a DHCP reservation. While you are at it you should

create DHCP reservations for all the IP devices that you plan to monitor, phones for example.

## STEPS TO GET INSTALLING RPI WITH DEVICEMON AND WEBSERVER SERVICES

Personally, I like to use Microsoft Remote Desktop but it's your choice. It's not required but it makes the installation and editing of the details in the Python script much easier.

***sudo apt-get install xrdp***

Gets everything up to snuff if not already done at install

***sudo apt-get update***

***sudo apt-get upgrade***

Gets rid of excess junk

***sudo apt autoremove***

Tools for XWindows Manipulation - these are used by the menu provided to set the size of the window but are not required.

***sudo apt-get install xdotool***

Copy over the services directory to /home/pi so the **full path is /home/pi/services**

Change permissions on shell scripts ***/home/pi/services/menu.sh*** and

***/home/pi/service/devicemon/devicescan.sh*** to be executable.

Bluetooth is probably already installed, but if not do this.

***sudo apt-get install bluetooth***

***sudo apt-get install bluez***

Install the Bluetooth libraries

***sudo apt-get install libbluetooth-dev***

Setup bluez for Python

***sudo apt-get install python-pip python-dev ipython***

***sudo pip install pybluez***

***sudo pip3 install pybluez***

## CONFIGURE THE SERVICES

This allows all of these to run in the background and the menu.sh makes it easy to stop\start\review each of the services without having to remember anything. This also ensures that the services will start automatically at boot time. These commands create a

hard link to a service file and then enable the service.

```
sudo ln /home/pi/services/devicemon/devicemon.service /etc/systemd/system  
sudo systemctl enable devicemon.service
```

```
sudo ln /home/pi/services/devicemon/btscan.service /etc/systemd/system  
sudo systemctl enable btscan.service
```

```
sudo ln /home/pi/services/devicemon/webserver.service /etc/systemd/system  
sudo systemctl enable webserver.service
```

## CONFIGURE PEOPLE AND DEVICES

Now you must configure the usernames, device names and device addresses (Bluetooth\BLE and IP). All this information is stored near the top of the script **/home/pi/services/devicemon/devicemon.py** and can be edited using the Thonny Python editor. You can look at the other settings for scan frequency, timeouts and retries but I'd suggest leaving as is until you have confirmed the basic operation. You will find a lot of information about the Bluetooth scanning options located inside of the devicemon.py script.

Using IP addresses is straightforward. If you have a DHCP reservation and get a consistent IP address this will work reliably if the device is available on the network.

Bluetooth is a little more nuanced so devicemon.py uses multiple methods for determining Bluetooth availability. These notes are located inside of the devicemon.py script as a reminder. For BT there are 3 different options:

- BT1 - Attempts direct connection to a specific BT address - Works for many devices but not all. Will work if device is paired.
- BT2 - Looks at bluetoothctl info for presence of a specific BT address. Deletes it if present and sees if it repopulates within 10 secs to indicate it is still present.
- BT3 - Looks at the last 4 minutes of Bluetooth extended info available via journalctl for a NAME match (data which is not available via "bluetoothctl devices". If present it extracts the related BT mac address and then tests its presence using the BT2 method. This is useful in the following situation. When the "Allow phone presence detection" is enabled in Smartthings settings the phone acts like a BLE beacon. However, in Android V6 and above the BT MAC address is virtualized\spoofed when in beacon mode so it is constantly changing the BT address so the name must be used.

**Tip:** Most of the work done in determining the presence of Bluetooth devices is performed by the built-in 'bluetoothctl' app. To see what BT devices are visible, type the following: bluetoothctl <enter> scan on < enter> and you will see the BT devices come and go. This is essentially what is happening in the btcscan.service.

## Testing BT Devices

The script devicescan.sh does a lot of the work and can be executed directly as shown in the screenshot below:

```
pi@raspberrypi:~/services $ cd devicemon
pi@raspberrypi:~/services/devicemon $ ./devicescan.sh IP 192.168.0.40 TRUE
Output has been enabled
Searching for IP Device matching: ipaddress
IP device found: 192.168.0.40
pi@raspberrypi:~/services/devicemon $ ./devicescan.sh BTA 4A:E6:9F:2E:33:91 TRUE
Output has been enabled
Searching for BT Device matching: 4A:E6:9F:2E:33:91
[DEL] Device 4A:E6:9F:2E:33:91 4A-E6-9F-2E-33-91
Device has been removed
Remaining Loops: 4
Remaining Loops: 3
Remaining Loops: 2
Remaining Loops: 1
BT device found: 4A:E6:9F:2E:33:91
pi@raspberrypi:~/services/devicemon $ ./devicescan.sh BTN GaryMoto TRUE
Output has been enabled
Found matching address 4A:E6:9F:2E:33:91 for name: GaryMoto
Searching for BT Device matching: 4A:E6:9F:2E:33:91
Device 4A:E6:9F:2E:33:91 not available
Remaining Loops: 4
Remaining Loops: 3
Remaining Loops: 2
Remaining Loops: 1
Remaining Loops: 0
Bluetooth device not found: 4A:E6:9F:2E:33:91
```

./devicescan.sh IP 192.168.0.40 TRUE (TRUE means show debug info)

./devicescan.sh BTA 4A:E6:9F:2E:33:91 TRUE (BTA means it is expecting a BT address)

./devicescan.sh BTN GaryMoto TRUE (BTN means it is expecting a partial name)

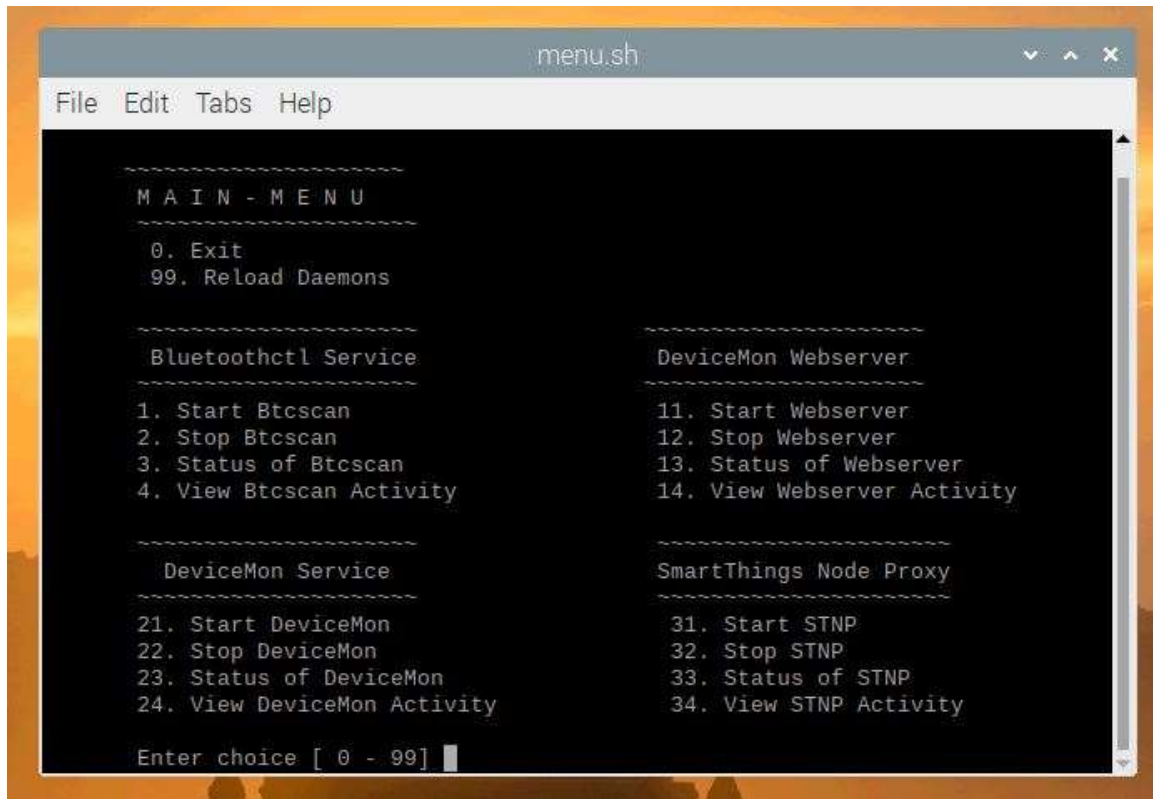
## IMPORTANT

Much of the grunt work is performed by devicescan.sh in parsing the output from bluetoothctl and journalctl. If you are running a different version of the bluez stack the parsing performed by devicescan.sh may not work correctly. Do not despair. It should be a simple edit to devicescan.sh to select a different field to return to devicemon.py.

## SIMPLE MENU SYSTEM

To make it easy to check on things I have created a simple menu system which you can start with:

***/home/pi/services/menu.sh***



```
menu.sh
File Edit Tabs Help

-----
M A I N - M E N U
-----
0. Exit
99. Reload Daemons

-----
Bluetoothctl Service
-----
1. Start Btscan
2. Stop Btscan
3. Status of Btscan
4. View Btscan Activity

-----
DeviceMon Service
-----
21. Start DeviceMon
22. Stop DeviceMon
23. Status of DeviceMon
24. View DeviceMon Activity

-----
DeviceMon Webserver
-----
11. Start Webserver
12. Stop Webserver
13. Status of Webserver
14. View Webserver Activity

-----
SmartThings Node Proxy
-----
31. Start STNP
32. Stop STNP
33. Status of STNP
34. View STNP Activity

Enter choice [ 0 - 99] █
```

You can ignore the options listed for 31-34. They will not do anything on your system and can be edited out of the menu if desired.

Now you can start each of the services. Watch for any errors as they start.

**Option 1** - Start Btscan. This starts bluetoothctl in scan mode so that BT and BLE signals are captured.

**Option 11** - Start Webserver. This starts a Python simple http server using port 8001 serving out the directory /home/pi/services/devicemon/webserver

**Option 21** - Start Devicemon. This starts the devicemon.py script and should generate the output file /home/pi/services/devicemon/webserver/unifiedpresence.json. Point a browser to your Raspberry Pi port 8001. For example: <http://192.168.0.110:8001> and you should see this file present. You can open the file and check its contents to make sure it is updating approximately once per minute.

#That is all that is required for now on the Pi device. You can check on the status of the three services using options 4, 14 and 24 of the menu and they should all be running. You can use the other menu options to view the status as well as the output of these services so you can check on them as needed. When viewing the service activity you can use PgDn for more then q and enter to exit. If you reboot the Pi all of these services will start automatically.

## SMARTTHINGS DEVICES

Now it's time to configure the SmartThings device. Locate the code for the Unified Presence device handler in this repository and then "Create New Device Handler" in your SmartThings account. Once the DH is created you can now create a new device using the Unified Presence device handler. Let's call it **"!Gary Presence"** for easy of reference.

On your SmartPhone open SmartThings Classic and go to your list of Things. Open up **"!Gary Presence"** and click on the gear in the upper right corner. Here you must enter the following information: IP address for your Raspberry Pi, port (8001) and the name of the person this Presence Monitor will represent (Bob). This name must exactly match one of the names you entered in the Python script (it is case sensitive). Change the Auto refresh for now to 1 minute for testing, you can change it back later.

The screen should now populate with the information in the unifiedpresence.json file. If the screen is not updating press the DNI: button which should force the device network interface to your specific IP\Port combination (in Hex).

The time alongside an In or Out status indicates the time at which a device transitioned from In to Out or Out to In status. For a person it is a combination of the underlying device states. One or more devices in, person In. All devices out, person Out.

You can now use this presence sensor in SmartThings in the normal way as a trigger for turning things off or on etc.

## SECONDARY DEVICES

Most homes have more than one person in them so it will often be desirable to have multiple Unified Presence devices in the same household. SmartThings requires that every device have a unique DNI (device network interface) which is combination of IP address and port represented in a hex format for IP devices. This is visible via the device properties in the Smartthings web portal. In order to create a secondary device, you must therefore have either a different IP or different port.

While you could have multiple webserver, each serving their own port it is easier to simply add additional IP addresses.

You can add additional temporary addresses as follows:

Edit the file: ***/home/pi/services/devicemon/webserver.sh***

This file is the startup script for the webserver as shown below.

```
File Edit Search Options Help
#!/bin/bash

#Add the additional ethernet interfaces if needed
ifconfig eth0:2 192.168.0.41 netmask 255.255.255.0 up &
ifconfig eth0:3 192.168.0.42 netmask 255.255.255.0 up &

# Or for WiFi with Pi Zero W or earlier Pi
#ifconfig wlan0:0 192.168.0.50 netmask 255.255.255.0 up
#ifconfig wlan0:1 192.168.0.51 netmask 255.255.255.0 up

#Start the Webserver
/usr/bin/python -m SimpleHTTPServer 8001
```

Comment in\out the addresses lines that you need.

These addresses are lost when the Pi is rebooted. It does not cause any issues if these same ifconfig commands are executed multiple times.

See here on how to add permanent secondary addresses if you choose but the above method is probably easier. <https://www.garron.me/en/linux/add-secondary-ip-linux.html>

You can now create additional Unified Presence devices for each family member. Just remember that each Unified Presence device must use a unique IP address.

## LIMITATIONS

This solution operates on a single RPi. Depending on the configuration of the house\apt there may not be an ideal placement that allows BT or BLE signals from any point in the house to reach the Pi. To supplement phone presence I also purchased a [BT 5.0 BLE beacon](#) to place inside the car. It met my needs of long battery life (2xAAA), strong signal (BT 5.0) and tunable, small (hockey puck) and cheap (\$20). Although it does not



have great reviews on Amazon, I'm using it in an entirely different way and it has been working great.

## NOTES

In my own testing I have found that different mobile devices respond quite differently, especially when not in use. A ping can always reach my Android phone, but my iPad goes to sleep and is mostly unreachable. However, my iPad does always respond to Bluetooth.

Some devices always respond to a directed "ping" of their BT address while others do not unless the device has been paired with the Pi.

If a device has the new version (not classic) version of SmartThings installed, they can enable "Allow phone presence detection" which is described as: "Allow other devices to know that your phone is nearby using BLE scanning". This should give it broader visibility to the RPi without pairing.

Although it is intended to represent an individual person there is nothing to prevent a Unified Presence device to mix devices from multiple people, his phone, his car, her phone, her car for example. The Unified Presence would thus indicate when both people were out.