The Raspberry Pi Zero 2 W
User Guide

Getting Started With the Raspberry Pi
Zero 2 and Zero 2

Samson Tobi
# Contents

Introduction ............................................................................................................. 1

CHAPTER ONE ...................................................................................................... 4
Hardware Setup .................................................................................................... 4

CHAPTER TWO .................................................................................................... 6
Installing the Raspberry Pi OS .............................................................................. 6

CHAPTER THREE .................................................................................................. 10
Using Raspbian .................................................................................................... 10

CHAPTER FOUR ................................................................................................... 13
Connecting to WiFi ............................................................................................... 13

CHAPTER FIVE ...................................................................................................... 14
Update Software ................................................................................................... 14

CHAPTER SIX ...................................................................................................... 17
Configuration ........................................................................................................ 17
Configuration of HDMI .......................................................................................... 66

CHAPTER SEVEN ................................................................................................. 72
The config.txt file ................................................................................................ 72
Raspberry Pi  HDMI Pipeline ............................................................................... 115

CHAPTER EIGHT .................................................................................................. 262
Troubleshooting ................................................................................................... 262

CHAPTER NINE .................................................................................................... 278
An Introduction to Python .................................................................................... 278
Introduction

The Raspberry Pi is a popular Single Board Computer (SBC) in that it is a full computer packed into a single board. Many may already familiar with the Raspberry Pi 4 and its predecessors, which comes in a form factor that has become as highly recognizable.

The Raspberry Pi comes in an even smaller form factor. The introduction of the Raspberry Pi Zero 2 allowed one to embed an entire computer in even smaller projects. This guide will cover the latest version of the Zero 2 product line, the Raspberry Pi Zero 2 2 - Wireless, which has an onboard WiFi module.

While these directions should work for most any version and form factor of the Raspberry Pi, it will revolve around the Pi Zero 2 W.

Required Materials
If you're looking for a starter pack, this kit includes everything you need to start using your Pi Zero 2 W.
Required Materials

To follow along with this tutorial, you will need the following items:

- Raspberry Pi Zero 2 W Basic Kit
- Monitor
- Keyboard
- Mouse (optional but suggested)
- USB hub (for more than one USB device)
CHAPTER ONE

Hardware Setup

Depending on your use case, setup for the Pi Zero 2 2 can be minimal, or it can be cumbersome due to smaller connectors on the Zero 2 2 and the adapters needed to connect standard devices such as mice, keyboards, and monitors.

Monitor

- To attach the Pi Zero 2 to a Monitor or TV that has an HDMI input, attach a miniHDMI to HDMI cable or adapter to the miniHDMI connector on the Pi Zero 2. Connect the other end to the HDMI port on your monitor or television.

- Connect the USB OTG cable to the Pi Zero 2 via the microUSB connector. If you have a keyboard/mouse combe, attach your dongle to the standard female USB end. If you have a separate mouse and keyboard, you will need a USB hub to attach both to the USB OTG cable.
• Make sure that you have a valid Raspberry Pi image on your microSD card (more on this later). Insert the microSD card into the microSD slot.

• Power your Pi Zero 2 via the microUSB power input.

There are a few other connectors to point out but we won't be using. The Pi Zero 2 has a 40 pin GPIO connector on the board that matches the pinout of the standard Pi 3. You can solder wires, headers or Pi Hats to this connector to access the GPIO pins or even power. The camera connector will allow you to connect the Raspberry Pi camera although it is worth noting that the connector is a 22pin 0.5mm and different than the standard Pi and will need a different cable to connect the camera to the Pi.
CHAPTER TWO

Installing the Raspberry Pi OS

When it comes to creating an image on an SD card for your Pi Zero 2 W, there are two options.

**Option 1: NOOBS**

The Raspberry Pi foundation has created NOOBS (New Out Of the Box Software) which is easy to use and get you started. Most generic kits such as the Pi3 Starter Kit will come with a NOOBS uSD card. Unfortunately as of the release of the Pi Zero 2 Basic Kit the NOOBS image will need to be updated to work on the Pi Zero 2 boards.

If you don't have a current NOOBS image or just want to start again from scratch installing NOOBS is easy, you don't need any special software to install it on a card. Just download the image, unzip, drag and drop the files to your card, and you are good to go.

Once you have NOOBS on your card, insert it in your board, apply power, and you should see action on your screen.
Follow the prompts and install Raspbian (you may also want to select a different region for your keyboard and language). You can also go into the Wifi screen and setup Wifi. If you do you will get a lot more options for images to install.

**Option 2: .img File**

If you want something other than the basic Raspbian install or other options found on NOOBS, you will need to install your own image on the uSD card.

This method is slightly more involved because you need a special *.img file that not only puts your files on the card, but also sets up things like making the card bootable.

The Raspberry Pi foundation has a handful of images like Ubuntu, OSMC (Open Source Media Center), and even Windows 10 IOT Core. A google search will find many more including specialized images for certain tasks. If you’ve never worked with Raspberry Pi before, we recommend Raspbian. You can download the latest version using the link below.

To install your own image on your card, we recommend software called Etcher.
These guys have taken all the different steps needed and put them all in one piece of software to take care of everything. Download your image, then run the program, select your image, select your uSD card drive, and then hit flash. Once it is done, remove your card and you are good to go. Once the image is installed, insert the card into the board and apply power.

For Mac users, the ApplePi Baker Software is a great way to upload a new image to an SD card. It will ask for an admin password upon startup. Select the SD card on the left plane, then upload your image under the Pi Ingrediants: IMG Recipe section.

Click Restore Backup, wait for the progress bar to finish, and you're done. The program even ejects the card you, so can yank it right out and insert it into your Pi.
For the rest of this tutorial, we'll assume you've installed Raspbian either by installing the image directly or with Noobs. The tutorial should also work fine for most Linux based systems with a Graphical User Interface, but things might be in slightly different locations
CHAPTER THREE

Using Raspbian

Now that you've gotten your board up and running let's go over some basics.

Raspbian and most Raspberry Pi OS are going to be Linux based. Don't let that scare you too much. Gone are the days of having to remember lots of commands or that you need to type :wq to save and exit your text editor.

Linux now a days has a Graphical User Interface (GUI) similar to Windows or MacOS, and, while you will probably want to learn a few basic commands and shortcunts, you can usually get away with not using them.

In the upper left hand corner of Raspbian you will see six icons. The first is a Raspberry. This is basically the same as your 'Start Menu' on Windows machines. Click on this, and you will see a menu of all the installed programs as well as the shutdown options.

The second icon looks like a globe, and it is the Web Browser. Then we have the "File Manager", "Terminal", "Mathematica", and finally "Wolfram".
Changing Your Password

An important thing to remember is that Linux has user names and passwords. The default Raspian user id is "pi" and the password is "raspberry". The second thing to remember is that I know your user id and password and so do lots of other people with less honorable intentions.

Don't assume all viruses are written for full sized computers. Raspberry Pis tend to have a reasonable amount of processing power and are often left running with very little to do. Recently viruses have been showing up that were written specifically to log into Raspberry Pis using the default user name and password and use their processing power to mine crypto currencies.

So, the first thing we are going to do is change the password. Open up a terminal window and type
pi@raspberrypi:~ $ sudo passwd

You will be prompted to enter your new password. Type in your new password, and your Pi will be that much more secure from unwanted visitors.
CHAPTER FOUR

Connecting to WiFi

Next, let's connect the Pi Zero 2 W to the internet. If you have a Pi Zero 2, you will need to add a Wifi dongle to your board's USB port (you may need a USB hub at this point). The Pi Zero 2 W has built in WiFi, so you will not need any external WiFi dongles.

To enable WiFi on your Pi, look at the upper right corner of the Desktop, and left click the WiFi icon. You should see a list of available networks. Select the one you want. If it is a secured network, it will prompt you for a password. Enter the network password, and press OK. Give it a few seconds, and you should be connected.

The WiFi symbol on the Desktop will change its appearance once connected. If you setup WiFi in Noobs, this information will already be saved. Adding WiFi isn't strictly necessary, but you will need internet access if you plan on doing updates.
CHAPTER FIVE

Update Software

The last thing we are going to do is to update all the software packages on the board. Packages are constantly updated, and no one wants to be left behind. Luckily, Linux uses a package manager. All we have to do is tell our package manager to update everything, and then sit back and watch. Let's go over a few of the commands we'll be using. Go ahead and open the terminal, if it isn't already open. Type the following.

pi@raspberrypi:~ $ sudo apt-get update:

This will go fetch the latest package information and tell the package manager what needs to be updated.

- **Sudo**: (also known as super user) is a command that you will see a lot, specifically with high security commands. It makes sure you have the correct privileges. Depending on settings, it may or may not ask you for a password.

- **apt-get**: is the package manager and **update** is the command we are giving it.
pi@raspberrypi:~ $ sudo apt-get upgrade:

This will download and upgrade all the packages. Please note that this upgrade will take a while. It will also prompt you if this is really what you want to do, the correct answer is "Y".

pi@raspberrypi:~ $ sudo shutdown -r now:

- **shutdown**: will shutdown the machine. `-r` tells it to reboot after shutting down, and `now` tells it to do so now (15 would tell the machine to shutdown in 15 minutes).

What happens if you don't type in sudo? Linux will give you an error saying you don't have permission to do something. Generally if you see that error stop for a second and think if you really know what you are doing and want to do it. If you do, then type sudo in front of your command, and try again.

**Other Useful Linux Commands**

A few other useful commands for use in the terminal command line:
- **pwd** - Print Working Directory, if you not sure what folder you are in this will tell you where you are in the filesystem.
- **ls** - List, this will show you the contents of the folder. To show all files, including hidden ones, type `ls -a` to show all files/folders. Alternatively, typing `ls -al` will show you all files/folders as well as their permission settings.
- **cd** - this is how you change directories. `cd foldername` will move you to that folder. `cd ..` will back you up one level. `cd ~` will take you back to your home directory.
- **passwd** - this will allow you to change your password
- **man** - this stands for manual. Type `man` before a command to get a summary of how to use it.
- **nano** - this will open a basic text editor that is fairly easy to use.

At this point you should be interacting with your Raspberry Pi like you would any other computer. You can teach yourself the finer points of Linux, learn Python, program the GPIO pins, setup a minecraft server, build a network storage system, game console, or media center, or just surf the web.
CHAPTER SIX

Configuration

The *raspi-config* is Alex Bradbury’s original Raspberry Pi configuration tool. To begin, type the following command on the command line:

```bash
sudo raspi-config
```

Sudo is required because you will be editing files that do not belong to you as the pi user.

**Note**

If you're using the Raspberry Pi desktop, you can configure your Raspberry Pi using the graphical Raspberry Pi Configuration application from the Preferences menu.

After that, you should see a blue screen with the following options in a grey box:
Note the menu depicted may vary slightly.

The up and down arrow keys are used to navigate the highlighted selection between the available options. By pressing the right arrow key, you can navigate away from the Options menu and to the Select> and Finish> buttons.

By pressing left, you'll return to the options. Alternatively, you can switch between these by pressing the Tab key.

In general, raspi-config is intended to provide functionality for performing the most common configuration changes. This may result in automated modifications to /boot/config.txt and other common Linux configuration files.
Certain configuration options require a reboot to take effect. If any of those have been modified, raspi-config will prompt you to reboot now when you click the Finish> button.

Note

Additionally, you can type a letter to skip to a particular section of a long list of option values (such as the list of time zone cities). For instance, entering L will take you directly to Lisbon, which is only two options away from London, avoiding the need to scroll all the way through the alphabet.

Note

Due to the ongoing development of the raspi-config tool, the list of available options below may be incomplete. Additionally, please keep in mind that different Raspberry Pi models may come with a variety of different options.

System Configuration

The system options submenu enables you to customize various aspects of the boot, login, and networking processes, as well as some other system-level settings.
Allows for the configuration of the wireless LAN’s SSID and passphrase.

Audio Specify the location of the audio output.

**Password**

On Raspberry Pi OS, the default user is pi with the password raspberry. This can be changed here. Peruse the profiles of other users.

**Hostname**

Set the Pi’s visible name on a network.

**Autologin / Boot**

This submenu allows you to choose whether to boot to console or desktop and whether or not to log in. If automatic login is selected, you will be logged in as the pi user.

**Networking During Boot**

This option allows you to delay booting until a network connection is established.

Enable or disable the splash screen that appears at boot time.
LEDs with high output

If your Pi model supports it, you can use this option to modify the behavior of the power LED.

Options for Display

Resolution

Define the default HDMI/DVI video resolution that the system will use when it boots without a connected TV or monitor. If the VNC option is enabled, this may have an effect on RealVNC.

Under scan

The size of the image produced by older television sets varied significantly; some had cabinets that overlapped the screen. Thus, television images were given a black border to ensure that no part of the image was lost; this is referred to as overscan.

Modern televisions and monitors do not require a border, and the signal does not support it. If the initial text displayed on the screen vanishes beyond the edge, you must enable overscan to restore the border.
Any modifications will take effect following a reboot. By editing config.txt, you can exert greater control over the settings.

Disabling overscan on some displays, particularly monitors, causes the image to fill the entire screen and corrects the resolution. Other displays may require leaving overscan enabled and adjusting its values.

**Doubling the Pixels**

Activate/deactivate 2x2 pixel mapping.

**Video Composite**

Enable composite video on the Raspberry Pi 4. Composite video is disabled by default on models prior to the Raspberry Pi4, and thus this option is not displayed.

**Enable or disable screen blanking.**

**Options for Interfacing**

There are the following options to enable/disable in this submenu: SSH, VNC, SPI, I2C, Serial, 1-wire, and Remote GPIO are all included.
Camera Configure the CSI camera interface by enabling/disabling it.

SSH Allow or deny remote command line access to your Raspberry Pi via SSH.

SSH enables remote access to the Raspberry Pi's command line from another computer. By default, SSH is disabled. More information about SSH can be found on the SSH documentation page. If you are connecting your Pi directly to a public network, you should disable SSH until all users have secure passwords.

VNC Configure the RealVNC virtual network computing server by enabling/disabling it.

SPI Enable/disable SPI interfaces and load the SPI kernel module automatically, which is required for products such as PiFace.

I2C Enable/disable I2C interfaces and load the I2C kernel module automatically.

Serial Configure the serial connection to enable/disable shell and kernel messages.
1-wire Activate or deactivate the Dallas 1-wire interface. Typically, this is used with DS18B20 temperature sensors.

**GPIOs located remotely**

Remote access to the GPIO pins can be enabled or disabled.

**Performance Alternatives**

**Overclock**

On certain models, this tool allows you to overclock your Raspberry Pi's CPU. The amount of overclocking you can achieve varies; excessive overclocking may result in instability. When this option is selected, the following warning appears:

Bear in mind that overclocking your Raspberry Pi may shorten its life. If a certain level of overclocking causes system instability, experiment with a lower level of overclocking. While the computer is booting, hold down the Shift key to temporarily disable overclocking.

**Memory on the GPU**

Increase or decrease the amount of memory available to the GPU.
File System with Overlay

Activate or deactivate a read-only file system

Fan

Configure the behavior of a fan connected via GPIO.

Alternatives for Localization

You can select from the following options in the localisation submenu: keyboard layout, time zone, locale, and wireless LAN country code.

Locale

Choose a locale, for instance en GB.

UTF-8 UTF-8UTF-8UTF-8UTF-8UTF-8UTF-8UTF-8

Zone hoarier

Select your local time zone by first selecting a region, such as Europe, and then a city, such as London. By typing a letter, you can navigate down the alphabetical list to that letter.
Keyboard

This option brings up another menu where you can choose your keyboard layout. It will take a long time to display while the keyboard is being read. Changes are usually immediately effective, but may necessitate a reboot.

Wireless LAN Country

This option configures your wireless network's country code.

Optional Extras

Extend your file system

This option will expand your installation to take up the entire SD card, freeing up additional space for files. This will require a reboot of the Raspberry Pi.

Warning

There is no confirmation: selecting the option immediately initiates the partition expansion.

GL Driver Configure the experimental GL desktop graphics drivers by enabling/disabling them.
GL (Full KMS) Enable/disable the experimental OpenGL desktop graphics driver with Full KMS (kernel mode setting).

GL (Fake KMS) Controls whether the experimental OpenGL Fake KMS desktop graphics driver is enabled or disabled.

Legacy

Activate/deactivate the legacy non-GL VideoCore desktop graphics driver.

Compositor Enable/Display the composition manager xcompmgr

Names of Network Interfaces

Predictable network interface names can be enabled or disabled.

Configuration of the Network Proxy

Configure the proxy settings for the network.
Order of Boots

On the Raspberry Pi 4, you can specify whether to boot from USB or the network in the absence of an SD card. Additional information is available on this page.

Version of the Boot loader

On the Raspberry Pi 4, you can instruct the system to use the most recent boot ROM software or to revert to the factory default if the most recent version causes issues.

Update

This tool should be updated to the most recent version.

About raspi-config

Selecting this option shows the following text:

Finish

When you are finished with your changes, click this button. You will be prompted to reboot or not. When a
device is used for the first time, it is recommended that it be rebooted. If you chose to resize your SD card, there will be a delay in rebooting.

**Network Configuration**

Within the Raspberry Pi Desktop, a GUI is provided for configuring wireless connections in Raspberry Pi OS. If you are not using the Raspberry Pi Desktop, however, you can configure wireless networking via the command line.

Using the Desktop The network icon at the right-hand end of the menu bar can be used to establish wireless connections. If you're using a Raspberry Pi with integrated wireless connectivity, or if you've plugged in a wireless dongle, left-clicking this icon will display a list of available wireless networks.

If no networks are discovered, the message 'No APs discovered - scanning...' will appear. Wait a few seconds without closing the menu, and the network should be discovered.

Note that until the country code is set on Raspberry Pi devices that support the 5GHz band (Pi3B+, Pi4, CM4,
Pi400), wireless networking is disabled for regulatory reasons. To configure the country code, open the Raspberry Pi Configuration application from the Preferences Menu, select Localisation, and enter the correct code.

The icons on the right indicate whether a network is secure and the strength of its signal. Select the network to which you wish to connect. If it is secured, you will be prompted to enter the network key in a dialogue box:
Please keep in mind that raspi-config does not include all of the necessary options for configuring wireless networking; you may need to refer to the additional sections below for additional information if raspi-config fails to connect the Pi to the requested network.

**Obtaining Wireless LAN Network Information**

To perform a wireless network scan, type `sudo iwlist wlan0 scan`. This will display a list of all available wireless networks, as well as other pertinent information. Keep an eye out for:

1. 'ESSID:"testing"' is the wireless network's name.

2. The authentication method is 'IE: IEEE 802.11i/WPA2 Version 1'. It's WPA2, the newer and more secure wireless standard that replaces WPA in this case. This guide should
be applicable to WPA or WPA2, but may not be applicable to WPA2 enterprise. Additionally, you'll require the wireless network's password. For the majority of home routers, this information is located on a sticker on the router's back. For the examples below, the ESSID (ssid) is testing and the password (psk) is testing Password.

**Adding the Network Details to your Raspberry Pi**

*Open the wpa-supplicant configuration file in nano:*

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Go to the bottom of the file and add the following:

```yaml
network={
    ssid="testing"
    psk="testingPassword"
}
```

The password can be specified as an ASCII string enclosed in quotes, as in the preceding example, or as a pre-encrypted 32-byte hexadecimal number. You can generate an encrypted PSK using the wpa pass utility. This generates the encrypted PSK using the SSID and password.
Using the preceding example, you can generate the PSK using wpa password "testing". Then you will be prompted for the wireless network's password (in this case testingPassword). The following is the output:

```
network={
    ssid="testing"
    #psk="testingPassword"
    psk=131e1e221f6e06e3911a2d11ff2fac9182665c004de65300f5cac208a6a80531
}
```

It is worth noting that the plain text version of the code is included but commented out. To increase security, you should delete this line from the final wpa supplicant file.

The wpa pass tool requires a password with a length of between eight and sixty-three characters. To use a more complex password, you can extract the contents of a text file and pass it to wpa passphrase as an argument. Save the password in a text file and pass it to wpa pass by invoking wpa pass with the argument wpa pass "testing" file where password is stored. For additional After encrypting the file where password is stored, you should delete it to ensure that no plain text copy of the original password remains on the system.
To use the encrypted PSK created by wpa pass, either copy and paste it into the wpa supplicant.conf file, or redirect the tool's output to the configuration file in one of two ways:

- Either log in as root using sudo su or run wpa pass "testing" >> /etc/wpa_supplicant/wpa_supplicant.conf and enter the testing password when prompted.

- Alternatively, use wpa pass "testing" | sudo tee -a /etc/wpa_supplicant/wpa_supplicant.conf > /dev/null and enter the testing password when prompted; the redirection to /dev/null prevents tee from printing to the screen as well (standard output).

If either of these two options is chosen, ensure that you use >> or -a with tee — both will append text to an existing file. When a single chevron > is used, or when -a is omitted when using tee, all contents are deleted and the output is appended to the specified file.

Now save the file by pressing Ctrl+X, followed by Y, and then Enter.
With `wpa cli -i wlan0 reconfigure`, reconfigure the interface.

You can verify that it connected successfully by running `ifconfig wlan0`. If the `inet addr` field contains an address, the Raspberry Pi has established a connection to the network. If this is not the case, verify that your password and ESSID are correct.

Additionally, you must configure the country code on the Raspberry Pi 3B+ and Raspberry Pi 4B so that the 5GHz networking can select the appropriate frequency bands. This can be accomplished through the `raspi-config` application: navigate to the 'Localisation Options' menu and then to 'Change Wi-Fi Country'. Alternatively, you can add the following to the `wpa_supplicant.conf` file. (Note: You must replace 'GB' with your country's two-letter ISO code. (For a list of two-letter ISO 3166-1 country codes, see Wikipedia.)
Nota bene, with the latest Buster Raspberry Pi OS release, you must ensure that the top of the wpa supplicant.conf file contains the following information:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=<Insert 2 letter ISO 3166-1 country code here>
```

**Utilization of Unsecured Networks**

If the network to which you are connecting does not require a password, the network's wpa supplicant entry must include the correct key mgmt entry. e.g.

```
network=
  ssid="testing"
  key_mgmt=NONE
```

**Warning**

While using unsecured wireless networks, extreme caution should be exercised.

Networks that are not visible
If you are using a hidden network, an additional option, scan ssid, in the wpa supplicant file may aid in connection.

```plaintext
network={
    ssid="yourHiddenSSID"
    scan_ssid=1
    psk="Your_wireless_network_password"
}
```

You can verify that it connected successfully by running `ifconfig wlan0`. If the `inet addr` field contains an address, the Raspberry Pi has established a connection to the network. If not, verify that your password and ESSID are valid.

**Configuring Multiple Wireless Networks**

On recent versions of Raspberry Pi OS, it is possible to configure wireless networking in multiple ways. For instance, you could have one at home and one at school.

For instance,
If you have a range of two networks, you can add a priority option to allow you to choose between them. The network with the highest priority within range will be connected.
the Dynamic Host Configuration Protocol (DHCP) Daemon

The Raspberry Pi configures TCP/IP across all of its network interfaces using dhcpcd. The dhcpcd daemon is designed to be a one-stop shop for ZeroConf configuration on UNIX-like systems. This procedure entails allocating an IP address to each interface, configuring netmasks, and configuring DNS resolution via the Name Service Switch (NSS) facility.

By default, Raspberry Pi OS attempts to configure all network interfaces automatically via DHCP, falling back to automatic private addresses in the range 169.254.0.0/16 in the event that DHCP fails. This behavior is consistent with that of other Linux distributions and Microsoft Windows.

IP Addresses That Remain Static

If you want to disable automatic configuration for an interface and instead configure it statically, edit `/etc/dhcpcd.conf` to include the necessary information. For instance:
**Network Configuration**

You must create a `wpa_supplicant.conf` file specific to your wireless network. Copy this file to the SD card's boot folder. When the Raspberry Pi first boots, it copies this file to the correct location in the Linux root file system and uses those settings to enable wireless networking.

Because the Raspberry Pi's IP address will not be visible immediately upon powering on, this step is critical for headless connectivity. Depending on the operating system and editor used to create this, the file may contain incorrect newlines or the incorrect file extension; therefore, use an editor that takes this into account.
Linux expects the newline character denoted by the line feed (LF).

**Warning**

After connecting your Raspberry Pi to power, allow a few minutes (up to 5) for it to boot up and register on the network.

*A wpa_supplicant.conf file example:*

```ini
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
country=<Insert 2 letter ISO 3166-1 country code here>
update_config=1

network={
    ssid="<Name of your wireless LAN>"
    psk="<Password for your wireless LAN>"
}
```

Note Without a keyboard or monitor, you'll need a way to access your headless Raspberry Pi remotely. SSH can be enabled for headless setup by placing a file named ssh, without any extension, in the SD Card's boot folder. For additional information, refer to the section on configuring an SSH server.
Configuring a Wireless Routed Access Point

A Raspberry Pi can be used as a wireless access point within an Ethernet network, thereby creating a secondary network. The Raspberry Pi manages the resulting new wireless network entirely.

Consider instead setting up a bridged access point to extend an existing Ethernet network to wireless clients.

A routed wireless access point can be created using the Raspberry Pi 4's, Raspberry Pi 3's, or Raspberry Pi Zero W's built-in wireless capabilities, or by using a suitable USB wireless dongle that supports access point mode.

Certain USB dongles may require minor configuration changes. Please consult the forums if you are having issues with a USB wireless dongle.
This documentation was tested on a Raspberry Pi 3B that was freshly installed with Raspberry Pi OS Buster.

**Prior to Commencement**

- Ascertain that you have administrative privileges on your Raspberry Pi. The network configuration will be modified during the installation process; it is recommended to have local access via a screen and keyboard connected to your Raspberry Pi.

- Connect your Raspberry Pi to your local network via Ethernet and boot the Raspberry Pi OS.

- Ensure that the Raspberry Pi OS is up to date on your Raspberry Pi and reboot if any packages were installed during the process.

- Take note of the Raspberry Pi’s Ethernet network’s IP configuration:
  1. For the purposes of this document, we will assume that Ethernet LAN IP network 10.10.0.0/24 is configured and that the Raspberry Pi will manage IP network 192.168.4.0/24 for wireless clients.
If your Ethernet LAN is already using IP network 192.168.4.0/24, please select another IP network for wireless, e.g. 192.168.10.0/24

• Be prepared to test your new access point with a wireless client (laptop, smartphone, etc.).

Installing Accounts Payable and Management Software

To function as an access point, the Raspberry Pi must be installed with the hostapd access point software package:

```
sudo apt install hostapd
```

Enable the wireless access point service and set it to start when your Raspberry Pi boots:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
```

In order to provide network management services (DNS, DHCP) to wireless clients, the Raspberry Pi needs to have the dnsmasq software package installed:

```
sudo apt install dnsmasq
```
Finally, install `netfilter-persistent` and its plugin `iptables-persistent`. This utility helps by saving firewall rules and restoring them when the Raspberry Pi boots:

```
sudo DEBIAN_FRONTEND=noninteractive apt install -y netfilter-persistent iptables-persistent
```

To configure the static IP address, edit the configuration file for dhcpcd with:

```
sudo nano /etc/dhcpcd.conf
```

Go to the end of the file and add the following:

```
interface wlan0
  static ip_address=192.168.4.1/24
  nohook wpa_supplicant
```

**Configure Routing and IP Masking**

This section configures the Raspberry Pi so that wireless clients can connect to computers connected to the main (Ethernet) network and then to the internet.
Note

If you want to prevent wireless clients from connecting to the Ethernet network or the internet, skip to the next section.

To enable routing, that is, to allow traffic to flow between networks on the Raspberry Pi, create a file with the following contents using the following command:

```
sudo nano /etc/sysctl.d/routed-ap.conf
```

File contents:

```
# Enable IPv4 routing
net.ipv4.ip_forward=1
```

By enabling routing, hosts on network 192.168.4.0/24 will be able to connect to the LAN and the main router to the internet. To allow traffic between clients on this foreign wireless network and the internet without changing the configuration of the main router, the Raspberry Pi can use a "masquerade" firewall rule to replace the IP address of wireless clients with its own LAN IP address.
All outgoing traffic from wireless clients will be interpreted as coming from the Raspberry Pi, enabling communication with the internet.

The Raspberry Pi will accept all incoming traffic, restore the original IP addresses, and forward it to the original wireless client.

*This process is configured by adding a single firewall rule in the Raspberry Pi:*

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Now save the current firewall rules for IPv4 (including the rule above) and IPv6 to be loaded at boot by the `netfilter-persistent` service:

```
sudo netfilter-persistent save
```

The directory `/etc/iptables` contains the filtering rules. If you change the configuration of your firewall in the future, make sure to save the changes before rebooting.
Configure the wireless network's DHCP and DNS services.

Dnsmasq provides DHCP and DNS services. The default configuration file acts as a template for all possible configuration options, of which we require only a few. It is simpler to begin with an empty file.

Replace the default configuration file with a new one:
Rename the default configuration file and edit a new one:

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
sudo nano /etc/dnsmasq.conf
```

Add the following to the file and save it:

```bash
interface=wlan0 # Listening interface
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
# Pool of IP addresses served via DHCP
domain=wlan # Local wireless DNS domain
address=/gw.wlan/192.168.4.1
# Alias for this router
```

The Raspberry Pi will provide wireless DHCP clients with IP addresses between 192.168.4.2 and 192.168.4.20 with a lease time of 24 hours. Wireless clients should be able to connect to the Raspberry Pi via the name gw.wlan.
There are numerous additional configuration options for dnsmasq; for details, see the default configuration file (/etc/dnsmasq.conf) or the online documentation.

Ascertain Wireless Functioning

Countries throughout the world regulate the use of radio frequency bands for telecommunications to ensure interference-free operation. The Linux operating system assists users in adhering to these rules by enabling applications to be configured with a two-letter "WiFi country code," for example, US for a computer used in the United States.

In the Raspberry Pi OS, 5 GHz wireless networking is disabled until the user configures a WiFi country code, which is typically done during the initial installation process (see wireless configuration pages in this section for details.)

To ensure that your Raspberry Pi's WiFi radio is not disabled, run the following command:

```bash
sudo rfkill unblock wlan
```
This setting will be restored automatically at boot time. Following that, we'll define a country code in the access point software configuration.

**Configure the Access Point (AP) Software**

Create the *hostapd configuration file*, located at `/etc/hostapd/hostapd.conf`, to add the various parameters for your new wireless network.

**`sudo nano /etc/hostapd/hostapd.conf`**

Include the following information in the configuration file. This configuration makes the assumption that we're using channel 7, with the network name NameOfNetwork and the password AardvarkBadgerHedgehog. Nota bene, the name and password should not be enclosed in quotes. The password should be between eight and sixty-four characters long.
Take note of the line `country_code=GB`: it tells the computer to use the appropriate wireless frequencies in the United Kingdom. Adapt this line to include your country's two-letter ISO code. A list of two-letter ISO 3166-1 country codes is available on Wikipedia.

To use the 5 GHz band, switch from `hw_mode=g` to `hw_mode=a`. `hw_mode` can take the following values:

- `a` = IEEE 802.11a (5 GHz) (from the Raspberry Pi 3B+)
- IEEE 802.11b = `b` (2.4 GHz)
- IEEE 802.11g = (2.4 GHz)

Take note that when changing the `hw_mode`, you may also need to change the channel - a list of allowed combinations is available on Wikipedia.
Utilizing the new Wireless Access Point

Restart your Raspberry Pi and check to see if the wireless access point becomes available automatically

`sudo systemctl reboot`

Before your Raspberry Pi has restarted, use your wireless client to search for available wireless networks. The network SSID that you specified in the file `/etc/hostapd/hostapd.conf` should now be present and accessible using the specified password.

If SSH is facilitated on the Raspberry Pi, you should be able to connect to it using the following method from your wireless client, assuming the `pi` account exists:

```
ssh pi@192.168.4.1 or ssh pi@gw.wlan ssh pi@192.168.4.1
```

If your wireless client has access to your Raspberry Pi (and, if you configured routing, to the internet), you have successfully configured your new access point!

If you run into difficulties, you can seek assistance in the forums. Kindly include a link to this page in your message.
Configuring a Wireless Access Point with a Bridged Architecture.

Within an existing Ethernet network, the Raspberry Pi can be used as a bridged wireless access point. This will enable the network to be expanded to include wireless computers and devices. Consider instead setting up a routed access point if you wish to create a standalone wireless network.

A bridged wireless access point can be created using the built-in wireless capabilities of the Raspberry Pi 4, 3, or Zero W, or by using a suitable USB wireless dongle that supports access point mode. Certain USB dongles may require minor configuration changes.

Please consult the forums if you are having issues with a USB wireless dongle.

This documentation was tested on a Raspberry Pi 3B that was freshly installed with Raspberry Pi OS Buster.
Prior to Commencement

Ascertain that you have administrative privileges on your Raspberry Pi. As part of the installation, the network configuration will be completely reset: local access via a screen and keyboard connected to your Raspberry Pi is recommended.

Note

If installing remotely via SSH, use the name of your Raspberry Pi rather than the IP address, e.g. ssh pi@raspberrypi.local, as the Raspberry Pi's network address will almost certainly change after installation. Additionally, you should be prepared to add a screen and keyboard if necessary in the event that you lose contact with your Raspberry Pi following installation. Connect your Raspberry Pi to your network via Ethernet and boot the Raspberry Pi operating system. Ascertain that the Raspberry Pi OS is up to date on your Raspberry Pi and reboot if packages were installed during the process. Prepare a wireless client (laptop, smartphone, etc.) to verify your new access point's functionality.
Installing Accounts Payable and Management Software

To function as a conjugated access point, the Raspberry Pi must be installed with the hosted access point software package: install hostapd with sudo apt install hostapd

Enable the wireless access point service and configure it to start immediately upon booting your Raspberry Pi:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
```

The software installation process has been completed. Later on, we will configure the access point software.

Construct a Network Bridge

Using the Raspberry Pi’s built-in interfaces, a bridge network device will connect the Ethernet and wireless networks.

Construct a bridge device and populate it

Create a file with the following contents to add a bridge network device named bro:

```
sudo nano /etc/systemd/network/bridge-bro.netdev
```

File contents:

```
[NetDev]
```
Name=br0
Kind=bridge

To connect the Ethernet network to the wireless network, create the following file to add the built-in Ethernet interface (eth0) as a bridge member:

```bash
sudo nano /etc/systemd/network/br0-member-eth0.network
```

File contents:

```
[Match]
Name=eth0

[Network]
Bridge=br0
```

Note

The access point software will add the wireless interface wlan0 to the bridge when the service starts. There is no need to create a file for that interface. This situation is particular to wireless LAN interfaces.

Now enable the `systemd-networkd` service to create and populate the bridge when your Raspberry Pi boots:

```bash
sudo systemctl enable systemd-networkd
```

Configure the bridge device's IP address
Members of a bridge device's network interfaces are never assigned an IP address, as they communicate via the bridge. The bridge device itself requires an IP address in order to communicate with the Raspberry Pi over the network.

On the Raspberry Pi, dhcpcd, the DHCP client, automatically requests an IP address for each active interface. Thus, we must disable the eth0 and wlan0 interfaces and allow dhcpcd to configure only br0 via DHCP.

```
sudo nano /etc/dhcpcd.conf
```

Add the following line near the beginning of the file (above the first interface xxx line, if any):

```
denyinterfaces wlan0 eth0
```

Add the following line near the file's start (above the first interface xxx line, if there is one):

```
wlan0 eth0 denyinterfaces
```

Add the following at the end of the file:
This line configures interface bro according to the DHCP defaults. Save the file to complete the machine's IP configuration.

**Ascertain Wireless Functioning**

Countries throughout the world regulate the use of radio frequency bands for telecommunications to ensure interference-free operation. The Linux operating system assists users in adhering to these rules by enabling applications to be configured with a two-letter "WiFi country code," for example, US for a computer used in the United States.

In the Raspberry Pi OS, 5 GHz wireless networking is disabled until the user configures a WiFi country code, which is typically done during the initial installation process (see wireless configuration pages in this section for details.)

To ensure that your Raspberry Pi’s WiFi radio is not disabled, run the following command:
**sudo rfkill wlan unblock**

This setting will be restored automatically at boot time. Following that, we'll define a country code in the access point software configuration.

**Configure the Access Point (AP) Software**

Add the various parameters for your new wireless network to the hostapd configuration file located at

```
/etc/hostapd/hostapd.conf
```

```
/etc/hostapd/hostapd.conf sudo nano
```

Include the following information in the configuration file. This configuration makes the assumption that we're using channel 7, with the network name NameOfNetwork and the password AardvarkBadgerHedgehog. Nota bene, the name and password should not be enclosed in quotes. The password should be between eight and sixty-four characters long.
Note the lines interface=wlan0 and bridge=br0: these direct hostapd to add the wlan0 interface as a bridge member to br0 when the access point starts, completing the bridge between Ethernet and wireless.

**Note**

The line country_code=GB: it configures the computer to use the correct wireless frequencies in the United Kingdom. Adapt this line and specify the two-letter ISO code of your country. See Wikipedia for a list of two-letter ISO 3166-1 country codes.

To use the 5 GHz band, you can change the operations mode from hw_mode=g to hw_mode=a. Possible values for hw_mode are:

```conf
country_code=GB
interface=wlan0
bridge=br0
ssid=NameOfNetwork
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=AardvarkBadgerHedgehog
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```
a = IEEE 802.11a (5 GHz) (Raspberry Pi 3B+ onwards)
b = IEEE 802.11b (2.4 GHz)
g = IEEE 802.11g (2.4 GHz)

Note that when changing the hw_mode, you may need to also change the channel - see Wikipedia for a list of allowed combinations.

Run the new Wireless AP
Now restart your Raspberry Pi and verify that the wireless access point becomes automatically available.

```
sudo systemctl reboot
```

Once your Raspberry Pi has restarted, search for wireless networks with your wireless client. The network SSID you specified in file `/etc/hostapd/hostapd.conf` should now be present, and it should be accessible with the specified password.
If your wireless client has access to the local network and the internet, congratulations on setting up your new access...
point! If you encounter difficulties, contact the forums for assistance. Please refer to this page in your message.

Using a Proxy Server
If you want your Raspberry Pi to access the Internet via a proxy server (perhaps from a school or other workplace), you will need to configure your Pi to use the server before you can get online.

You will require the following:
- Your proxy server's IP address or hostname and port
- Your proxy's username and password (if required)

Configuration of the Raspberry Pi
Three environment variables (http proxy, https proxy, and no proxy) must be configured to instruct the Raspberry Pi how to access the proxy server.

Open a terminal window, and open the file /etc/environment using **nano**:

```
sudo nano /etc/environment
```

To create the http proxy variable, add the following to the /etc/environment file:
http proxy="http://proxyipaddress:proxyport" export http
proxy="http://proxyipaddress:proxyport"

Replace *proxyipaddress and proxyport* with your proxy's
IP address and port.

Keep in mind that if your proxy requires a username and
password, you must enter them in the following format:

http
proxy="http://username:password@proxyipaddress:proxyp
ort" export
http
proxy="http://username:password@proxyipaddress:proxyipo
rt"

Export the environment variable https proxy with the same
information:

export https proxy="http://username:password@proxyipaddress:proxypo
rt"
Create the environment variable no proxy, which contains a comma-separated list of addresses for which your Pi should not use the proxy:

\texttt{no proxy=}

"127.0.0.1, localhost"

Your /etc/environment file should now have the following structure:

\begin{verbatim}
export http_proxy="http://username:password@proxyipaddress:proxyport"
export https_proxy="http://username:password@proxyipaddress:proxyport"
export no_proxy="localhost, 127.0.0.1"
\end{verbatim}

To save and exit, press Ctrl + X.

Ensure that the sudoers file is up to date.
You must update sudoers in order for operations that run as sudo (e.g., downloading and installing software) to use the new environment variables.

To open sudoers, run the following command:

`sudo visudo`

Add the following line to the file so `sudo` will use the environment variables you just created:

```
Defaults   env_keep+="http_proxy https_proxy no_proxy"
```

To save and exit, press Ctrl + X.

Restart the Raspberry Pi

Boot up your Raspberry Pi to apply the changes. You should now be able to use your proxy server to access the internet.
Configuration of HDMI

In the overwhelming majority of instances, simply connecting your HDMI-equipped monitor to the Raspberry Pi via a standard HDMI cable will result in the Pi automatically selecting the highest resolution supported by the monitor.

Due to the Raspberry Pi Zero’s use of a mini HDMI port, you will require a mini HDMI to full-size HDMI cable or adapter.

Due to the Raspberry Pi 4’s two micro HDMI ports, you'll need one or two micro HDMI to full-size HDMI leads or adapters, depending on the number of displays you intend to connect. Before powering on the Raspberry Pi, you should connect any HDMI leads.

The Raspberry Pi 4 can power up to two displays with a maximum resolution of 1080p and a refresh rate of 60Hz. When two displays are connected at 4K resolution, the refresh rate is limited to 30Hz.

Additionally, you can drive a single 4K display at a 60Hz refresh rate; this requires connecting the display to the HDMI port adjacent to the USB-C power input (labelled HDMI0).
Additionally, you must enable 4Kp60 output in config.txt by setting the hdmi enable 4kp60=1 flag. This flag can also be set via the desktop environment's 'Raspberry Pi Configuration' tool.

If you are using the 3D graphics driver (also known as the FKMS driver), you will find a graphical application for configuring standard displays, including multi-display configurations, in the Preferences menu.

**Note**
To save and exit, press Ctrl + X.

**Restart the Raspberry Pi**
The Screen Configuration tool (arandr) is a graphical interface for configuring multiple displays and selecting display modes. This tool is available in the desktop Preferences menu, but only if the 3D graphics driver is installed, as this driver provides the necessary mode setting functionality.
Select the screen, resolution, and orientation using the Configure menu option. Drag the displays around to any position you want if you're using a multi-screen setup.
When you've completed the necessary configuration, click the Tick button to save the changes.

If you are using legacy graphics drivers, or find yourself in a situation where the Raspberry Pi is unable to determine the optimal mode, or if you wish to set a resolution other than the default, the remainder of this page may be useful.

**Note**
All commands are fully documented in the documentation's config.txt section.

**Groups and Modes of HDMI**
HDMI is divided into two groups: CEA (Consumer Electronics Association, which oversees the standard that is typically used by televisions) and DMT (Display Monitor Timings, the standard typically used by monitors). Each group promotes a distinct set of modes, each of which describes the output's resolution, frame rate, clock rate, and aspect ratio.

**Which Modes is My Device Compatible With?**
You can use the tvservice command line application to determine which modes your device supports, as well as other useful information:
tvservice -s returns information about the current HDMI status, including the mode and resolution.

Tvservice -m CEA displays a list of all CEA modes that are supported.

Tvservice -m DMT displays a list of all DMT modes that are currently supported.

If you're using a Pi 4 with multiple displays connected, tvservice must be informed which device to query for information. You can obtain display IDs for all attached devices by using the following command:

```
tvservice -l tvservice -l tvservice -l
```

You can specify which display tvservice uses by prefixing the tvservice command with `-v display id>`, for example:

```
tvservice -v 7 -m CEA returns a list of all CEA modes supported for display ID 7.
```

**Configuring a Custom HDMI Mode**

Setting a specific mode is accomplished via the `config.txt` entries `hdmi group` and `hdmi mode`. The group entry determines whether to use CEA or DMT, while the mode setting determines the resolution and frame rate. Although tables of supported modes are included on the `config.txt`
Video Configuration page, you should use the tvservice command described above to determine which modes your device supports precisely.

To specify the HDMI port on the Raspberry Pi 4, append an index identifier to the hdmi group or hdmi mode entry in config.txt, for example, hdmi mode:0 or hdmi group:1.

**Customizing the HDMI Mode**

There are two ways to configure a custom mode: via hdmi cvt or via hdmi timings.

hdmi cvt creates a new Coordinated Video Timing entry:

**Configuration of the Video**

Occasionally, it may be necessary to define the precise clock requirements for the HDMI signal. This is a completely customizable mode that is enabled by setting hdmi group=2 and hdmi mode=87.

Then, using the hdmi timings config.txt command, you can configure the display's specific parameters. hdmi timings specifies all of the timings required by an HDMI signal. Typically, these timings can be found on the datasheet for the display in use.
hdmi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch> <h_sync_pulse> <h_back_porch>

<table>
<thead>
<tr>
<th>Timing</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>h_active_pixels</td>
<td>The horizontal resolution</td>
</tr>
<tr>
<td>h_sync_polarity</td>
<td>0 or 1 to define the horizontal sync polarity</td>
</tr>
<tr>
<td>h_front_porch</td>
<td>Number of horizontal front porch pixels</td>
</tr>
<tr>
<td>h_sync_pulse</td>
<td>Width of horizontal sync pulse</td>
</tr>
<tr>
<td>h_back_porch</td>
<td>Number of horizontal back porch pixels</td>
</tr>
<tr>
<td>v_active_lines</td>
<td>The vertical resolution</td>
</tr>
<tr>
<td>v_sync_polarity</td>
<td>0 or 1 to define the vertical sync polarity</td>
</tr>
<tr>
<td>v_front_porch</td>
<td>Number of vertical front porch pixels</td>
</tr>
<tr>
<td>v_sync_pulse</td>
<td>Width of vertical sync pulse</td>
</tr>
<tr>
<td>v_back_porch</td>
<td>Number of vertical back porch pixels</td>
</tr>
<tr>
<td>v_sync_offset_a</td>
<td>Leave at 0</td>
</tr>
<tr>
<td>v_sync_offset_b</td>
<td>Leave at 0</td>
</tr>
<tr>
<td>pixel_rep</td>
<td>Leave at 0</td>
</tr>
<tr>
<td>frame_rate</td>
<td>Frame rate of mode</td>
</tr>
<tr>
<td>interlaced</td>
<td>0 for non-interlaced, 1 for interlaced</td>
</tr>
<tr>
<td>pixel_freq</td>
<td>The mode pixel frequency</td>
</tr>
<tr>
<td>aspect_ratio</td>
<td>The aspect ratio required</td>
</tr>
</tbody>
</table>

aspect_ratio should be one of the following:
CHAPTER SEVEN

The config.txt file

What is config.txt?

Instead of the BIOS that you'd find on a traditional PC, the Raspberry Pi uses a configuration file. The system configuration parameters, that would normally be modified and saved using a BIOS, are instead saved in a text file called config.txt. Before the ARM CPU and Linux are initialized, the GPU reads this.

It, together with bootcode.bin and start.elf, must be stored in the first (boot) partition on your SD card. From Linux, this file is generally accessed as /boot/config.txt and must be modified as root. It appears as a file in the only accessible part of the card in Windows or OS X. If you need to use some of the configuration options listed below but don't have a config.txt file on your boot disk yet, simply create one.

After you've rebooted your Raspberry Pi, any modifications will take effect. After Linux has booted, use the following commands to see the current active settings:
- **vcgencmd get_config <config>**: this displays a specific config value, e.g. *vcgencmd get_config arm_freq*.
- **vcgencmd get_config int**: this lists all the integer config options that are set (non-zero).
- **vcgencmd get_config str**: this lists all the string config options that are set (non-null).

**Note**

Some configuration settings are not retrievable with *vcgencmd*.

**File Format**

Because the initial boot firmware reads the *config.txt* file, it has a relatively simple file format. Each line has a single *property=value* statement, with *value* being either an integer or a string. By beginning a line with the # character, you can add comments or comment out and disable existing config values. Entries are limited to 80 characters per line; any characters beyond this limit will be ignored.

Here's a file that you can use as an example:

```
# Force the monitor to HDMI mode so that sound will be sent over HDMI cable
```
hdmi_drive=2

# Set monitor mode to DMT

hdmi_group=2

# Set monitor resolution to 1024x768 XGA 60Hz (HDMI_DMT_XGA_60)

hdmi_mode=16

# Make display smaller to stop text spilling off the screen

overscan_left=20

overscan_right=12

overscan_top=10

overscan_bottom=10

Memory Options

gpu_mem

The amount of memory to set aside for the GPU's exclusive use in megabytes; the remaining memory is allotted to the ARM CPU for use by the OS.
The default value for Pis with less than 1GB of memory is 64; for Pis with 1GB or more of memory, the default value is 76. The GPU's RAM is used for display, 3D, codec, and camera functions, as well as some basic firmware maintenance. The maximums shown below presume that you're using all of these options.

If you aren't, smaller \textit{gpu mem} numbers can be used.

Set \textit{gpu mem} to the smallest number feasible to get the best performance out of Linux. If a specific graphics feature isn't working, consider raising the value of \textit{gpu mem}, keeping in mind the recommended maximums shown below.

\textbf{Important}

Unlike GPUs found on x86 machines, where increasing memory can enhance 3D performance, the VideoCore's architecture means that specifying values bigger than necessary has no performance benefit, and can even hurt performance. The 3D component of the GPU on the Raspberry Pi 4 has its own memory management unit (MMU) and does not use the \textit{gpu mem} allocation. Instead, memory in Linux is allocated dynamically. In comparison
to prior models, this allows a smaller number for \textit{gpu mem} to be given on the Pi 4.

The following are the recommended maximum values:

<table>
<thead>
<tr>
<th>total RAM</th>
<th>\textit{gpu mem} recommended maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>256MB</td>
<td>128</td>
</tr>
<tr>
<td>512MB</td>
<td>384</td>
</tr>
<tr>
<td>1GB or greater</td>
<td>512, 256 on the Pi4</td>
</tr>
</tbody>
</table>

Setting \textit{gpu mem} to bigger values is feasible, but it should be avoided because it can cause issues, such as stopping Linux from booting. Although the minimal setting is 16, this disables some GPU functions. You can also use \textit{gpu mem 256}, \textit{gpu mem 512}, and \textit{gpu mem 1024} to swap SD cards between Pis with varying amounts of RAM without having to update config.txt every time:

\texttt{gpu\_mem\_256}

The \texttt{gpu\_mem\_256} For Raspberry Pis with 256MB of RAM, this command sets the GPU memory in megabytes. (If the memory capacity is less than 256MB, it is disregarded.) This takes precedence over \textit{gpu mem}. 
The **gpu_mem_512** For Raspberry Pis with 512MB of RAM, this command sets the GPU memory in megabytes. (If the memory capacity is less than 512MB, it is disregarded.) This takes precedence over **gpu mem**.

The **gpu_mem_1024** For Raspberry Pis with 1GB or greater memory, this command sets the GPU memory in megabytes. (If the memory size is less than 1GB, it is disregarded.) This takes precedence over **gpu mem**.

The **total_mem**

This parameter can be used to force a Raspberry Pi to limit its memory capacity: specify the total amount of RAM, in megabytes, you wish the Raspberry Pi to use. For example, to make a 4GB Raspberry Pi 4B behave as though it were a 1GB model, use the following:

**total_mem=1024**

This number will be limited to 128MB minimum and the total RAM installed on the board maximum.
disable_l2cache

Setting this to 1 prevents the CPU from accessing the GPU's L2 cache and necessitates the use of an L2 disabled kernel. On the BCM2835, the default value is 0.

Because the ARMs on the BCM2836, BCM2837, and BCM2711 have their own L2 cache, the default is 1. This variation in cache setting is reflected in the standard Pi kernel.img and kernel7.img builds.

Licence Key and Codec Options

Additional codecs can be decoded in hardware on the Pi 3 and earlier models by purchasing a license that is tied to your Raspberry Pi's CPU serial number. The hardware codecs for MPEG2 and VC1 are permanently disabled on the Raspberry Pi 4 and cannot be enabled even with a license key; however, MPEG2 and VC1 can be decoded in software via applications such as VLC on the Pi 4, thanks to its increased processing power over earlier models. As a result, if you're using a Pi 4, you won't need a hardware codec license key.
decode_MPG2

`decode_MPG2` is a hardware license key that allows you to use certain devices. MPEG-2 decoding, e.g. `decode_MPG2=0x12345678`.

decode_WVC1

`decode_WVC1` is a hardware license key that allows you to use certain devices. VC-1 decoding, e.g. `decode_WVC1=0x12345678`.

One can list up to eight license keys in a single `config.txt` file if you have many Raspberry Pis and have purchased a codec license for each of them, for example. `decode_MPG2=0x12345678,0xabcdabcd,0x87654321`. This allows you to change SD cards between Pis without having to update the `config.txt` file each time.

Video Options

Composite Video Mode

`sdtv_mode`

The `sdtv_mode` command specifies the composite video output TV standard. Composite video is output on the RCA connection on the first Raspberry Pi. Except for the Pi Zero
and Compute Module, all other Raspberry Pis output composite video and sound over the 4 pole TRRS ("headphone") connection. There is an unpopulated header designated "TV" on the Pi Zero that outputs composite video. Composite video is provided on the Compute Module through the TVDAC pin.

**sdtv mode** is set to 0 by default.

<table>
<thead>
<tr>
<th>sdtv_mode</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal NTSC</td>
</tr>
<tr>
<td>1</td>
<td>Japanese version of NTSC — no pedestal</td>
</tr>
<tr>
<td>2</td>
<td>Normal PAL</td>
</tr>
<tr>
<td>3</td>
<td>Brazilian version of PAL — 525/60 rather than 625/50, different subcarrier</td>
</tr>
<tr>
<td>16</td>
<td>Progressive scan NTSC</td>
</tr>
<tr>
<td>18</td>
<td>Progressive scan PAL</td>
</tr>
</tbody>
</table>

**sdtv_aspect**

The aspect ratio for composite video output is defined by the **sdtv aspect** command. 1 is the default value.

<table>
<thead>
<tr>
<th>sdtv_aspect</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4:3</td>
</tr>
<tr>
<td>2</td>
<td>14:9</td>
</tr>
<tr>
<td>3</td>
<td>16:9</td>
</tr>
</tbody>
</table>

**sdtv_disable_colourburst**
When `sdtv disable colorburst` is set to 1, colorburst on composite video output is disabled. Although the image will be monochrome, it may look sharper.

**enable_tvout (Raspberry Pi 4, Model B Only)**

Due to the way the internal clocks are interconnected and allocated on the Raspberry Pi 4, composite output is disabled by default.

Because composite video requires a very precise clock, putting that clock to the needed speed on the Pi 4 has a negative impact on other clocks attached to it, slowing down the entire system slightly.

We decided to disable composite video by default to avoid this system slowness because it is a less widely used feature. Use the `enable tvout=1` option to enable composite output. As previously stated, this will have a minor negative impact on performance. The composite behavior is the same on previous Pi devices.
HDMI Mode

Note

Most HDMI instructions can be used to any of the two HDMI ports on the Raspberry Pi 4B. To specify which port the setting must apply to, use the syntax `command>:port>`, where port is 0 or 1. The default port is 0 if no port is specified. The port is disregarded if you supply a port number on a command that doesn't require one. The HDMI sub-part of the conditionals portion of the documentation has more information on the syntax and alternative techniques.

The Raspberry Pi 4 features improved video hardware to allow dual 4k monitors, imposing modest constraints on the modes supported.

hdmi_safe

When `hdmi_safe` is set to 1, "safe mode" settings are utilized to try to boot with the most HDMI compatibility possible. Setting the following terms has the same effect:

`hdmi_force_hotplug=1`

`hdmi_ignore_edid=0xa5000080`
config_hdmi_boost=4

hdmi_group=2

hdmi_mode=4

disable_overscan=0

overscan_left=24

overscan_right=24

overscan_top=24

overscan_bottom=24

hdmi_ignore_edid

If the display does not have a correct EDID, setting `hdmi_ignore_edid` to `0xa5000080` allows you to ignore the EDID/display data. This uncommon value is required to make sure that it is not mistakenly triggered.

hdmi_edid_file

When `hdmi_edid_file` is set to 1, the GPU will read EDID data from the `edid.dat` file on the boot partition, rather than from the monitor. On the boards, you can get more information.
hdmiedid_filename

The **hdmiedid_filename** command on the Raspberry Pi 4B can be used to define the filename of the EDID file to use, as well as which port the file should be applied to. **hdmiedidfile=1** is also required to allow EDID files.

Consider the following scenario:

**hdmiedid_file=1**

**hdmiedid_filename:o=FileForPortZero.edid**

**hdmiedid_filename:i=FileForPortOne.edid**

hdmiedid_force_edid_audio

Setting **hdmiedid_force_edid_audio** to 1 makes the display pretend to accept all audio formats, allowing DTS/AC3 passthrough even if it isn't shown as supported.

hdmiedid_ignore_edid_audio

When **hdmiedid_ignore_edid_audio** is set to 1, the display thinks that all audio formats are unsupported. As a result, ALSA will use the analogue audio (headphone) jack by default.

hdmiedid_force_edid_3d
Setting `hmi force edid 3d` to 1 makes it appear as though all CEA modes support 3D, even if the EDID doesn't reflect it.

**hmi Ignore cec init**

If you set `hmi ignore cec init` to 1, the first active source message will not be sent during bootup. When you reboot your Raspberry Pi, this stops a CEC-enabled TV from coming out of standby and channel-switching.

**hmi Ignore cec**

When `hmi ignore cec` is set to 1, the TV pretends that CEC isn't supported at all. There will be no support for CEC functions.

**cec osd name**

The `cec osd name` command establishes the device's initial CEC name. Raspberry Pi is the default.

**hmi pixel encoding**

The pixel encoding mode is forced with the `hmi pixel encoding` command. Anything will use the mode requested by the EDID by default, so you shouldn't have to change it.
hdmi_max_pixel_freq

The firmware and KMS use the pixel frequency to filter HDMI modes. This is not to be confused with the frame rate. It specifies the maximum frequency that a legal mode can have, thus excluding modes with higher frequencies. If you want to disable all 4K modes, for example, you may give a maximum frequency of 200000000, because all 4K modes have frequencies greater than this.

hdmi_blanking

Whenever the operating system requests that the display be put into standby mode using DPMS to save power, the **hdmi bluffing** command determines what happens. The HDMI output is blanked but not switched off if this option is not specified or set to 0. Set this option to 1 to turn off the HDMI output as well, mimicking the behavior of other computers: the attached display will go into a low-power standby mode.
Note

Because this feature has not yet been enabled on the Raspberry Pi 4, setting `hdmi blanking=1` will not turn off the HDMI output. When running apps that don't use the framebuffer, such as `omxplayer`, this functionality may cause problems.

<table>
<thead>
<tr>
<th>hdmi_blanking</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HDMI output will be blanked</td>
</tr>
<tr>
<td>1</td>
<td>HDMI output will be switched off and blanked</td>
</tr>
</tbody>
</table>

**hdmi_drive**

You can switch from HDMI and DVI output modes with the `hdmi drive` command.

<table>
<thead>
<tr>
<th>hdmi_drive</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal DVI mode (no sound)</td>
</tr>
<tr>
<td>2</td>
<td>Normal HDMI mode (sound will be sent if supported and enabled)</td>
</tr>
</tbody>
</table>

**config_hdmi_boost**

Configures the HDMI interface's signal strength.

The minimum and maximum values are 0 and 11, respectively. For the original Model B and A, the default setting is 2. The Model B+ and all subsequent variants have
a default value of 5. If you're having problems with HDMI (speckling, interference), try setting it to 7.

Long HDMI cables may require up to 11, however these numbers should only be used if absolutely necessary. On the Raspberry Pi 4, this option is disabled.

**hdmi_group**

The *hdmi group* command specifies whether the HDMI output group is CEA (Consumer Electronics Association, the standard commonly used by TVs) or DMT (Digital Media Technology Association) (Display Monitor Timings, the standard typically used by monitors). *hdmi mode* should be used in combination with this setting.

<table>
<thead>
<tr>
<th>hdmi_group</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Auto-detect from EDID</td>
</tr>
<tr>
<td>1</td>
<td>CEA</td>
</tr>
<tr>
<td>2</td>
<td>DMT</td>
</tr>
</tbody>
</table>

**hdmi_mode**

*hdmi mode* defines the HDMI output format in conjunction with *hdmi group*. The CTA specification is used to generate the format mode numbers. See the forums
for further information on how to set a custom display mode that isn't included here.

Note

On some models, not all modes are available. If \textit{hdmi group=1} (CEA), the following settings are valid:

<table>
<thead>
<tr>
<th>\textit{hdmi mode}</th>
<th>Resolution</th>
<th>Frequency</th>
<th>Screen Aspect</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VGA (640x480)</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>480p</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>480p</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>720p</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1080i</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>480i</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>480i</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>240p</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>240p</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>480i</td>
<td>60 Hz</td>
<td>4:3</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>11</td>
<td>480i</td>
<td>60 Hz</td>
<td>16:9</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>12</td>
<td>240p</td>
<td>60 Hz</td>
<td>4:3</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>13</td>
<td>240p</td>
<td>60 Hz</td>
<td>16:9</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>14</td>
<td>480p</td>
<td>60 Hz</td>
<td>4:3</td>
<td>pixel doubling</td>
</tr>
<tr>
<td>15</td>
<td>480p</td>
<td>60 Hz</td>
<td>16:9</td>
<td>pixel doubling</td>
</tr>
<tr>
<td>16</td>
<td>1080p</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>576p</td>
<td>50 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>576p</td>
<td>50 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>720p</td>
<td>50 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1080i</td>
<td>50 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>576i</td>
<td>50 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>576i</td>
<td>50 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>288p</td>
<td>50 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>288p</td>
<td>50 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>576i</td>
<td>50 Hz</td>
<td>4:3</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>26</td>
<td>576i</td>
<td>50 Hz</td>
<td>16:9</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>27</td>
<td>288p</td>
<td>50 Hz</td>
<td>4:3</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>28</td>
<td>288p</td>
<td>50 Hz</td>
<td>16:9</td>
<td>pixel quadrupling</td>
</tr>
<tr>
<td>29</td>
<td>576p</td>
<td>50 Hz</td>
<td>4:3</td>
<td>pixel doubling</td>
</tr>
<tr>
<td>30</td>
<td>576p</td>
<td>50 Hz</td>
<td>16:9</td>
<td>pixel doubling</td>
</tr>
<tr>
<td></td>
<td>1080p</td>
<td>50Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1080p</td>
<td>30Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1080p</td>
<td>24Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1080p</td>
<td>25Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>1080p</td>
<td>30Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>480p</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>480p</td>
<td>60Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>576p</td>
<td>50Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>576p</td>
<td>50Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>1080i</td>
<td>50Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1080i</td>
<td>100Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>720p</td>
<td>100Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>576p</td>
<td>100Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>576p</td>
<td>100Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>576i</td>
<td>100Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>576i</td>
<td>100Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>1080i</td>
<td>120Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>720p</td>
<td>120Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>480p</td>
<td>120Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>480p</td>
<td>120Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>480i</td>
<td>120Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>480i</td>
<td>120Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>576p</td>
<td>200Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>576p</td>
<td>200Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>576i</td>
<td>200Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>576i</td>
<td>200Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>480p</td>
<td>240Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>480p</td>
<td>240Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>480i</td>
<td>240Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>480i</td>
<td>240Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>720p</td>
<td>24Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>720p</td>
<td>25Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>720p</td>
<td>30Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>1080p</td>
<td>120Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1080p</td>
<td>100Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Custom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>720p</td>
<td>25Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>67</td>
<td>720p</td>
<td>30Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>68</td>
<td>720p</td>
<td>50Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>69</td>
<td>720p</td>
<td>60Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>70</td>
<td>720p</td>
<td>100Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>71</td>
<td>720p</td>
<td>120Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>72</td>
<td>1080p</td>
<td>24Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>73</td>
<td>1080p</td>
<td>25Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>74</td>
<td>1080p</td>
<td>30Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>75</td>
<td>1080p</td>
<td>50Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>76</td>
<td>1080p</td>
<td>60Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>77</td>
<td>1080p</td>
<td>100Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>78</td>
<td>1080p</td>
<td>120Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>79</td>
<td>1680x720</td>
<td>24Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>80</td>
<td>1680x720</td>
<td>25Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>81</td>
<td>1680x720</td>
<td>30Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>82</td>
<td>1680x720</td>
<td>50Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>83</td>
<td>1680x720</td>
<td>60Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>84</td>
<td>1680x720</td>
<td>100Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>85</td>
<td>1680x720</td>
<td>120Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>86</td>
<td>2560x720</td>
<td>24Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>87</td>
<td>2560x720</td>
<td>25Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>88</td>
<td>2560x720</td>
<td>30Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>89</td>
<td>2560x720</td>
<td>50Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>90</td>
<td>2560x720</td>
<td>60Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>91</td>
<td>2560x720</td>
<td>100Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>92</td>
<td>2560x720</td>
<td>120Hz</td>
<td>64:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>93</td>
<td>2160p</td>
<td>24Hz</td>
<td>16:9</td>
<td>Pi 4</td>
</tr>
<tr>
<td>94</td>
<td>2160p</td>
<td>25Hz</td>
<td>16:9</td>
<td>Pi 4</td>
</tr>
</tbody>
</table>
Pixel doubling and quadrupling suggest a faster clock rate, as each pixel is replicated twice or four times. If `hdmi group=2` (DMT), the following settings are valid:

<table>
<thead>
<tr>
<th>Pixel</th>
<th>Resolution</th>
<th>Refresh Rate</th>
<th>Aspect Ratio</th>
<th>HDMI Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>2160p</td>
<td>30Hz</td>
<td>16:9</td>
<td>Pi 4</td>
</tr>
<tr>
<td>96</td>
<td>2160p</td>
<td>50Hz</td>
<td>16:9</td>
<td>Pi 4</td>
</tr>
<tr>
<td>97</td>
<td>2160p</td>
<td>60Hz</td>
<td>16:9</td>
<td>Pi 4</td>
</tr>
<tr>
<td>98</td>
<td>4096x2160</td>
<td>24Hz</td>
<td>256:135</td>
<td>Pi 4</td>
</tr>
<tr>
<td>99</td>
<td>4096x2160</td>
<td>25Hz</td>
<td>256:135</td>
<td>Pi 4</td>
</tr>
<tr>
<td>100</td>
<td>4096x2160</td>
<td>30Hz</td>
<td>256:135</td>
<td>Pi 4</td>
</tr>
<tr>
<td>101</td>
<td>4096x2160</td>
<td>50Hz</td>
<td>256:135</td>
<td>Pi 4</td>
</tr>
<tr>
<td>102</td>
<td>4096x2160</td>
<td>60Hz</td>
<td>256:135</td>
<td>Pi 4</td>
</tr>
<tr>
<td>103</td>
<td>2160p</td>
<td>24Hz</td>
<td>54:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>104</td>
<td>2160p</td>
<td>30Hz</td>
<td>54:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>105</td>
<td>2160p</td>
<td>30Hz</td>
<td>54:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>106</td>
<td>2160p</td>
<td>50Hz</td>
<td>54:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>107</td>
<td>2160p</td>
<td>60Hz</td>
<td>54:27</td>
<td>Pi 4</td>
</tr>
<tr>
<td>HDMI_mode</td>
<td>Resolution</td>
<td>Frequency</td>
<td>Screen Aspect</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>1</td>
<td>640x480</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>640x480</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>720x480</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>800x600</td>
<td>72Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td>incompatible with the Raspberry Pi</td>
</tr>
<tr>
<td>16</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1024x768</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1152x864</td>
<td>60Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1280x768</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>23</td>
<td>1280x768</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>24</td>
<td>1280x768</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>25</td>
<td>1280x768</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>26</td>
<td>1280x768</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>27</td>
<td>1280x800</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>28</td>
<td>1280x800</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>29</td>
<td>1280x800</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>30</td>
<td>1280x800</td>
<td>60Hz</td>
<td>15:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>No.</td>
<td>Resolution</td>
<td>Frequency</td>
<td>Aspect Ratio</td>
<td>Feature</td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>-----------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>31</td>
<td>1280x800</td>
<td>120 Hz</td>
<td>16:10</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>32</td>
<td>1280x860</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1280x960</td>
<td>85 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>1280x960</td>
<td>120 Hz</td>
<td>4:3</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>35</td>
<td>1280x1024</td>
<td>60 Hz</td>
<td>5:4</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>1280x1024</td>
<td>75 Hz</td>
<td>5:4</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>1280x1024</td>
<td>85 Hz</td>
<td>5:4</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>1280x1024</td>
<td>120 Hz</td>
<td>5:4</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>39</td>
<td>1360x768</td>
<td>60 Hz</td>
<td>16:9</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1360x768</td>
<td>120 Hz</td>
<td>16:9</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>41</td>
<td>1400x1050</td>
<td>60 Hz</td>
<td>4:3</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>42</td>
<td>1400x1050</td>
<td>75 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>1400x1050</td>
<td>85 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>1400x1050</td>
<td>120 Hz</td>
<td>4:3</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>45</td>
<td>1440x900</td>
<td>60 Hz</td>
<td>16:10</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>46</td>
<td>1440x900</td>
<td>75 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>1440x900</td>
<td>85 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>1440x900</td>
<td>120 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>1440x900</td>
<td>120 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1440x900</td>
<td>120 Hz</td>
<td>16:10</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>51</td>
<td>1600x1200</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>1600x1200</td>
<td>65 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>1600x1200</td>
<td>70 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>1600x1200</td>
<td>75 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>1600x1200</td>
<td>85 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>1600x1200</td>
<td>120 Hz</td>
<td>4:3</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>1680x1050</td>
<td>60 Hz</td>
<td>16:10</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>58</td>
<td>1680x1050</td>
<td>75 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>1680x1050</td>
<td>85 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>1680x1050</td>
<td>120 Hz</td>
<td>16:10</td>
<td>reduced blanking</td>
</tr>
<tr>
<td>61</td>
<td>1680x1050</td>
<td>120 Hz</td>
<td>16:10</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1792x1344</td>
<td>60 Hz</td>
<td>4:3</td>
<td></td>
</tr>
</tbody>
</table>
Note

A pixel clock limit exists.

Prior to the Raspberry Pi 4, the highest supported mode was 1920x1200 at 60Hz with reduced blanking, however the Raspberry Pi 4 can handle up to 4096x2160 (also known as 4k) at 60Hz. Also, if you use both of the Raspberry Pi 4's HDMI ports for 4k output, you'll be limited to 30Hz on both.
hdomim_timings

This allows you to set raw HDMI timing values for a custom mode using the **hdmi group=2** and **hdmi mode=87** parameters.

```
hdmi_timings=<h_active_pixels>    <h_sync_polarity>
<h_front_porch>       <h_sync_pulse>    <h_back_porch>
<v_active_lines>      <v_sync_polarity>  <v_front_porch>
<v_sync_pulse>       <v_back_porch>    <v_sync_offset_a>
<v_sync_offset_b>    <pixel_rep>       <frame_rate>
<intertaced>          <pixel_freq>     <aspect_ratio>
```

- `<h_active_pixels>` = horizontal pixels (width)
- `<h_sync_polarity>` = invert hsync polarity
- `<h_front_porch>` = horizontal forward padding from DE active edge
- `<h_sync_pulse>` = hsync pulse width in pixel clocks
- `<h_back_porch>` = vertical back padding from DE active edge
- `<v_active_lines>` = vertical pixels height (lines)
- `<v_sync_polarity>` = invert vsync polarity
\(<v_{\text{front\_porch}}\) = vertical forward padding from DE active edge

\(<v_{\text{sync\_pulse}}\) = vsync pulse width in pixel clocks

\(<v_{\text{back\_porch}}\) = vertical back padding from DE active edge

\(<v_{\text{sync\_offset\_a}}\) = leave at zero

\(<v_{\text{sync\_offset\_b}}\) = leave at zero

\(<\text{pixel\_rep}\) = leave at zero

\(<\text{frame\_rate}\) = screen refresh rate in Hz

\(<\text{interlaced}\) = leave at zero

\(<\text{pixel\_freq}\) = clock frequency

\(\text{width} \times \text{height} \times \text{framerate}\)

\(<\text{aspect\_ratio}\) = *

* One of eight aspect ratios can be selected (pick the one that is closest to your screen):

\(\text{HDMI\_ASPECT\_4\_3} = 1\)

\(\text{HDMI\_ASPECT\_14\_9} = 2\)
HDMI_ASPECT_16_9 = 3
HDMI_ASPECT_5_4 = 4
HDMI_ASPECT_16_10 = 5
HDMI_ASPECT_15_9 = 6
HDMI_ASPECT_21_9 = 7
HDMI_ASPECT_64_27 = 8

hdmi_force_mode

Setting this value to 1 will eliminate all other modes from the internal list save those provided by \textit{hdmi mode} and \textit{hdmi group}, meaning it will not show in any numbered lists of modes. If a display appears to be ignoring the \textit{hdmi mode} and \textit{hdmi group} settings, this option may be useful.

edid_content_type

The EDID content type is forced to a certain value.

The options are:

- $0 =$ \textit{EDIDContentType_NODATA}, content type none.
- $1 =$ \textit{EDIDContentType_Graphics}, content type graphics, ITC must be set to 1
2 = EDID_ContentType_Photo, content type photo
3 = EDID_ContentType_Cinema, content type cinema
4 = EDID_ContentType_Game, content type game

hdmi_enable_4kp60 (Raspberry Pi 4, Model B Only)

Once connected to a 4K monitor, the Raspberry Pi 4B selects a 30hz refresh rate by default. This option allows you to choose between 60Hz and 120Hz refresh rates. Note that this will raise the Raspberry Pi’s power usage and temperature. It is not possible to use both micro HDMI ports to output 4Kp60 at the same time.

Which Values are Valid for my Monitor?

Only a few formats may be supported by your HDMI monitor. Use the following method for finding out which formats are supported:

1. Boot up your Raspberry Pi and set the output format to VGA 60Hz (hdmi group=1 and hdmi mode=1).
2. To get a list of CEA-supported modes, type the following command: /opt/vc/bin/tvservice -m CEA
3. To get a list of DMT-supported modes, type the following command: 
   `/opt/vc/bin/tvservice -m DMT`

4. To see your current state, type the following command:
   `/opt/vc/bin/tvservice -s`

5. Use the commands below to get more specific info from your monitor:
   `/opt/vc/bin/tvservice -d edid.dat;`  
   `/opt/vc/bin/edidparser edid.dat`

When diagnosing issues with the default HDMI mode, the `edid.dat` must also be provided.

**Custom Mode**

If your monitor needs a mode that isn't listed in the table above, you can create a custom CVT mode for it instead:

```
hdmi_cvt=<width> <height> <framerate> <aspect> <margins> <interlace> <rb>
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>(required)</td>
<td>width in pixels</td>
</tr>
<tr>
<td>height</td>
<td>(required)</td>
<td>height in pixels</td>
</tr>
<tr>
<td>framerate</td>
<td>(required)</td>
<td>framerate in Hz</td>
</tr>
<tr>
<td>margins</td>
<td>0</td>
<td>0=margins disabled, 1=margins enabled</td>
</tr>
<tr>
<td>interlace</td>
<td>0</td>
<td>0=progressive, 1=interlaced</td>
</tr>
<tr>
<td>rb</td>
<td>0</td>
<td>0=normal, 1=reduced blanking</td>
</tr>
</tbody>
</table>
Using the default settings, fields towards the end might be omitted. It's worth noting that this merely establishes the mode (group 2 mode 87). You'll need to modify some more settings to make the Pi use this by default. For example, the following command specifies a resolution of 800 480 and enables audio drive:

`hdmi_cvt=800 480 60 6`

`hdmi_group=2`

`hdmi_mode=87`

`hdmi_drive=2`

If your display does not support normal CVT timings, this may not function.

**LCD Displays and Touchscreens**

`ignore_lcd`

When the Raspberry Pi LCD display is identified on the I2C bus, it is used by default. Because `ignore_lcd=1` skips the detection phase, the LCD display will not be used.

`display_default_lcd`
If a Raspberry Pi DSI LCD is detected, it will be utilized as the default display, and the framebuffer will be displayed.

The LCD will not be the default display if `display default lcd=0` is set, which normally means the HDMI output will be the default. The LCD can still be utilized by selecting it from a list of supported programs, such as omxplayer.

**lcd_framerate**

In Hertz/fps, define the framerate of the Raspberry Pi LCD display. The frequency is set to 60Hz by default.

**lcd_rotate**

It flips the display using the LCD's built-in flip functionality, which is a less expensive alternative to the GPU-based rotate operation.

For example, `lcd rotate=2` will adjust for a display that is turned upside down.

**disable_touchscreen**

Turn on or off the touchscreen.

`disable touchscreen=1` turns off the touchscreen on the Raspberry Pi’s official LCD display.
enable_dpi_lcd

Activate the LCD displays connected to the DPI GPIOs. This is to allow the parallel display interface to be used with third-party LCD displays.

dpi_group, dpi_mode, dpi_output_format

The config.txt parameters dpi group and dpi mode are used to set either predetermined modes or custom modes (DMT or CEA modes as used by HDMI above). Custom modes can be created in the same way that HDMI custom modes can (see the dpi timings section).

dpi_output_format is a bitmask that specifies the display format's numerous parameters.

dpi_timings

This allows you to set raw DPI timing parameters for a custom mode using the dpi group=2 and dpi mode=87 parameters.

dpi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch> <h_sync_pulse> <h_back_porch> <v_active_lines> <v_sync_polarity> <v_front_porch> <v_sync_pulse> <v_back_porch> <v_sync_offset_a>
\[ v\_sync\_offset\_b \quad \text{<pixel\_rep>} \quad \text{<frame\_rate>} \]
\[ \text{<interlaced>} \quad \text{<pixel\_freq>} \quad \text{<aspect\_ratio>} \]

\[ h\_active\_pixels = \text{horizontal pixels (width)} \]

\[ h\_sync\_polarity = \text{invert hsync polarity} \]

\[ h\_front\_porch = \text{horizontal forward padding from DE active edge} \]

\[ h\_sync\_pulse = \text{hsync pulse width in pixel clocks} \]

\[ h\_back\_porch = \text{vertical back padding from DE active edge} \]

\[ v\_active\_lines = \text{vertical pixels height (lines)} \]

\[ v\_sync\_polarity = \text{invert vsync polarity} \]

\[ v\_front\_porch = \text{vertical forward padding from DE active edge} \]

\[ v\_sync\_pulse = \text{vsync pulse width in pixel clocks} \]

\[ v\_back\_porch = \text{vertical back padding from DE active edge} \]

\[ v\_sync\_offset\_a = \text{leave at zero} \]

\[ v\_sync\_offset\_b = \text{leave at zero} \]
<pixel_rep> = leave at zero

<frame_rate> = screen refresh rate in Hz

<interlaced> = leave at zero

<pixel_freq> = clock frequency

(width*height*framerate)

<aspect_ratio> = *

* You can choose from eight different aspect ratios (the one that is closest to your screen):

HDMI_ASPECT_4_3 = 1

HDMI_ASPECT_14_9 = 2

HDMI_ASPECT_16_9 = 3

HDMI_ASPECT_5_4 = 4

HDMI_ASPECT_16_10 = 5

HDMI_ASPECT_15_9 = 6

HDMI_ASPECT_21_9 = 7

HDMI_ASPECT_64_27 = 8
Generic Display Options

**hdmi_force_hotplug**

Setting *hdmi force hotplug* to 1 makes it appear as if the HDMI hotplug signal is asserted and an HDMI display is connected. In other words, even if no HDMI monitor is identified, HDMI output mode will be used.

**hdmi_ignore_hotplug**

When *hdmi ignore hotplug* is set to 1, the HDMI hotplug signal is not asserted, making it look as though no HDMI display is connected. In other words, even if an HDMI monitor is recognized, composite output mode is used.

**disable_overscan**

Disable the *default overscan* values given by the firmware by setting disable overscan to 1. For HD CEA modes, the default overscan value for the left, right, top, and bottom borders is 48, 32 for SD CEA modes, and 0 for DMT modes. *disable overscan* is set to 0 by default.

**Note**

After this option, any extra overscan options, such as *overscan scale* or overscan edges, can still be used.
overscan_left

The \textit{overscan left} command specifies how many pixels to add to the firmware's default overscan setting on the screen's left side. \textit{o} is the default value.

If the text flows off the left side of the screen, increase this value; if the left side of the screen and the text are separated by a black border, decrease it.

overscan_right

On the right edge of the screen, the \textit{overscan right} command specifies the number of pixels to add to the firmware default value of overscan. \textit{o} is the default value.

If the text flow off the right edge of the screen, increase this value; if the right edge of the screen and the text are separated by a black border, decrease it.

overscan_top

The \textit{overscan top} command specifies how many pixels to add to the firmware's default overscan setting on the screen's top edge. \textit{o} is the default value.
If the text flows off the top edge of the screen, increase this amount; if the top edge of the screen and the text are separated by a black border, decrease it.

**overscan_bottom**

The *overscan bottom* command specifies how many pixels to add to the firmware's default overscan setting on the screen's bottom edge. 0 is the default value.

If the text flows off the bottom edge of the screen, increase this amount; if the bottom edge of the screen and the text are separated by a black border, decrease it.

**overscan_scale**

To require non-framebuffer layers to adhere to the overscan settings, set *overscan scale* to 1. 0 is the default value.

**NOTE:** This function is generally not advised because it can degrade image quality because the GPU will scale all layers on the display. To avoid images being scaled twice, it's best to disable overscan on the display itself (by the GPU and the display).
framebuffer_width

The console **framebuffer width** is specified in pixels with the framebuffer width command. The display width minus the total horizontal overscan is the default.

framebuffer_height

The console **framebuffer height** is specified in pixels with the framebuffer height command. The display height minus the total vertical overscan is the default.

max_framebuffer_height, max_framebuffer_width

The maximum dimensions of the internal frame buffer are specified here.

framebuffer_depth

To set the console **framebuffer depth** in bits per pixel, use framebuffer depth. 16 is the default value.

<table>
<thead>
<tr>
<th>framebuffer_depth</th>
<th>result</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8bit framebuffer</td>
<td>Default RGB palette makes screen unreadable</td>
</tr>
<tr>
<td>16</td>
<td>16bit framebuffer</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>24bit framebuffer</td>
<td>May result in a corrupted display</td>
</tr>
<tr>
<td>32</td>
<td>32bit framebuffer</td>
<td>May need to be used in conjunction with framebuffer_ignore_alpha=1</td>
</tr>
</tbody>
</table>
**framebuffer_ignore_alpha**

To disable the alpha channel, set `framebuffer_ignore alpha` to 1. Can aid with the presentation of a framebuffer depth of 32 bits.

**framebuffer_priority**

When running the traditional (pre-KMS) graphics driver on a system with multiple displays, this requires a specified internal display device to be the first Linux framebuffer. (i.e. `/dev/fbo`).

The options that can be set are:

<table>
<thead>
<tr>
<th>Display</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main LCD</td>
<td>0</td>
</tr>
<tr>
<td>Secondary LCD</td>
<td>1</td>
</tr>
<tr>
<td>HDMI 0</td>
<td>2</td>
</tr>
<tr>
<td>Composite</td>
<td>3</td>
</tr>
<tr>
<td>HDMI 1</td>
<td>7</td>
</tr>
</tbody>
</table>

**max_framebuffers**
The largest number of firmware framebuffers which can be produced is set by this configuration entry. The options 0, 1, and 2 are all valid.

When using more than one display, for example HDMI and a DSI or DPI display, this is set to 1 by default on devices before the Pi4. It will need to be increased to 2 when using so much than one display, for example HDMI and a DSI or DPI display.

Because the Raspberry Pi4 has two HDMI ports, the default setting is 2. In most circumstances, setting this to 2 is safe, because framebuffers are only created when an associated device is actually recognized. When utilized in headless mode, setting this value to 0 will prevent any framebuffers from being allocated, reducing memory needs.

**test_mode**

During boot, the **test mode** command displays a test image and sound for the specified number of seconds (through the composite video and analogue audio outputs only), before proceeding to load the OS normally. The default value is 0 and is used as a manufacturing test.
display_hDMI_rotate

To rotate or flip the HDMI display orientation, use `display_hDMI_rotate`.

0 is the default value.

<table>
<thead>
<tr>
<th>display_hDMI_rotate</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no rotation</td>
</tr>
<tr>
<td>1</td>
<td>rotate 90 degrees clockwise</td>
</tr>
<tr>
<td>2</td>
<td>rotate 180 degrees clockwise</td>
</tr>
<tr>
<td>3</td>
<td>rotate 270 degrees clockwise</td>
</tr>
<tr>
<td>0x10000</td>
<td>horizontal flip</td>
</tr>
<tr>
<td>0x20000</td>
<td>vertical flip</td>
</tr>
</tbody>
</table>

The 90 and 270 degree rotation choices both require additional GPU capacity, therefore they won't work with the 16MB GPU split. 90 and 270 degree rotations are not allowed when using the VC4 FKMS V3D driver (which is the default on the Raspberry Pi 4). For this driver, the Screen Configuration utility provides display rotations.
display_lcd_rotate

To rotate or flip the LCD orientation in the legacy graphics driver (default on models prior to the Pi4), use \textit{display lcd rotate}. The parameters for display hdmi rotate are the same as for \textit{display hdmi rotate}. Also see \textit{lcd rotate}.

display_rotate

In the newest firmware, \textit{display rotate} is deprecated, however it is kept for backwards compatibility. Instead, \textit{display lcd rotate} and \textit{display hdmi rotate} should be used. To rotate or flip the screen orientation, use \textit{display rotate}. The parameters are the same as \textit{display hdmi rotate}.

disable_fw_kms_setup

The firmware parses the EDID of any HDMI-connected display by default, selects an appropriate video mode, and then transmits the mode's resolution and frame rate, as well as overscan parameters, to the Linux kernel via kernel command line settings. In rare cases, this can result in the device selecting a mode that is not in the EDID and may be incompatible with it. To avoid this problem, set \texttt{disable fw kms setup=1} to prevent these parameters from being
passed. After that, the Linux video mode system (KMS) will analyze the EDID and choose an appropriate mode.

Other Options

**dispmanx_offline**

Offline *dispmanx* composition in two offscreen framebuffers is required.

This allows for the composite of more *dispmanx* elements, but it is slower and might even limit the screen framerate to 30 frames per second.
Raspberry Pi HDMI Pipeline

The HDMI composition pipeline hardware on the Raspberry Pi has been improved in a number of ways to allow twin screens and modes up to 4k60. One of the most significant modifications is that each clock cycle generates two output pixels. Every HDMI mode contains a set of timings that control all of the parameters related to the duration of sync pulses.

For each horizontal and vertical direction, these are normally characterized by a pixel clock, followed by a number of active pixels, a front porch, sync pulse, and back porch. Running everything at 2 pixel per clock means the Pi4 won't be able to handle any horizontal timings that aren't divisible by 2. Any mode that does not meet these requirements will be filtered out by the firmware and Linux kernel. There is just one mode that falls within this category under the CEA and DMT standards: DMT mode 81, which is 1366x768 @ 60Hz.

The horizontal sync and back porch timings in this mode have strange values.

It's also a unique mode in that its breadth isn't divisible by eight. If your monitor has this resolution, the Pi4 will
automatically switch to the next mode that the monitor advertises, which is usually 1280x720. Some monitors can be set to display 1360x768 pixels at 60 frames per second. They don't usually advertise this mode via their EDID, thus it can't be selected automatically, but it may be done manually by adding

\[ \text{hdmi\_group=2} \]

\[ \text{hdmi\_mode=87} \]

\[ \text{hdmi\_cvt=1360\ 768\ 60} \]

to config.txt.

Manually supplied \texttt{hdmi timings=} lines in \texttt{config.txt} must likewise adhere to the requirement that all horizontal timing parameters be divisible by 2. \texttt{dpi timings=} are not restricted in the same way because the pipeline still only runs at a single pixel per clock cycle.

**Onboard Analogue Audio (3.5mm Jack)**

Configure the onboard audio output to change the way analogue audio is driven and whether or not certain firmware features are enabled.

\[ \text{disable\_audio\_dither} \]
If the audio stream is routed to the analogue audio output, a 1.0LSB dither is applied by default. In some cases, such as when the ALSA volume is set to a low level, this can result in audible background "hiss." Disable the dither application by setting `disable audio dither` to 1.

**enable_audio_dither**

Whenever the audio samples are larger than 16 bits, audio dither (see `disable audio dither` above) is generally deactivated. Configure this option to 1 to force all bit depths to employ dithering.

**pwm_sample_bits**

The `pwm sample bits` command changes the analogue audio output's bit depth. The bit depth is set to 11 by default. Bit depths lower than 8 result in a PLL frequency that is too low to support, resulting in nonfunctional audio. This is primarily relevant as an example of how bit depth impacts quantization noise.
Camera Settings

disable_camera_led

When you set disable camera led to 1, the red camera LED will not turn on when you record video or take a still photo. When the camera is facing a window, for example, this is beneficial for preventing reflections.

awb_auto_is_greyworld

Setting awb auto is greyworld to 1 allows NoIR cameras to record legitimate photos and movies even if their libraries or programs don't support the greyworld option internally. It uses the "greyworld" algorithm instead of the "auto" awb mode. This should only be used with NoIR cameras or when the IR filter on a High Quality camera has been removed.

Boot Options

start_file, fixup_file

The firmware files delivered to the VideoCore GPU prior to booting are specified by these settings. The VideoCore firmware file to use is specified by start file. The fixup file parameter indicates the file that will be used to match the
GPU memory split in the start file. It's important to note that the start file and fixup file are a matched pair; otherwise, the board will not boot. Because this is an advanced option, we recommend that you use start x and start debug instead.

start_x, start_debug

These are the recommended approaches for picking firmware configurations since they provide a shortcut to some alternative start file and fixup file options.

start_x=1 implies

\[
\text{start\_file=start\_x.elf} \\
\text{fixup\_file=fixup\_x.dat}
\]

If the files start4x.elf and fixup4x.dat are present on the Raspberry Pi 4, they will be utilized instead.

start_debug=1 implies

\[
\text{start\_file=start\_db.elf} \\
\text{fixup\_file=fixup\_db.dat}
\]
When utilizing the camera module, `start x=1` should be given. This is automatically configured when you enable the camera with `raspi-config`.

**disable_commandline_tags**

Set the `disable_commandline_tags` command to 1 to stop `start.elf` before launching the kernel, by filling in ATAGS (memory from 0x100).

**cmdline**

`cmdline` is an alternate filename for reading the kernel command line string from the boot partition; the default value is `cmdline.txt`.

**kernel**

`kernel` is the alternate filename to use when loading the kernel from the boot partition. `Kernel.img` is the default value on the Pi 1, Pi Zero, and Compute Module, and `kernel7.img` is the default value on the Pi 2, Pi 3, and Compute Module 3. It's `kernel7l.img` on the Pi4.

**arm_64bit**

If non-zero, requires the kernel loading system to presume a 64-bit kernel, starts the processors in 64-bit mode, and
loads *kernel8.img* as the kernel image, unless an explicit *kernel* option is given, in which case the kernel image loaded is kernel8.img. On all platforms, the value is set to 0.

**NOTE:** Uncompressed image files or a gzip archive of an image can be used as 64-bit kernels (which can still be called kernel8.img; the bootloader will recognize the archive from the signature bytes at the beginning). The 64-bit kernel will only function with the newest firmware for the Pi4, Pi3, and Pi2B rev1.2 devices.

**arm_control**

**Warning**

This option has been deprecated; instead, use *arm 64bit* to enable 64-bit kernels. Sets the control bits for the board.

**armstub**

*armstub* is the name of the file on the boot partition from which the ARM stub will be loaded. The default ARM stub is saved in firmware and is automatically picked based on the Pi model and other variables. The stub is a short ARM program that runs before the kernel.
Its function is to configure low-level hardware, such as the interrupt controller, before handing control over to the kernel.

**arm_peri_high**

To activate "High Peripheral" mode on the Pi 4, set `arm_peri_high` to 1. If a sufficient DTB is loaded, it is set automatically.

**Note**

Your system will fail to boot if you enable "High Peripheral" mode without a compatible device tree. Because ARM stub support is currently unavailable, you'll need to use `armstub` to load an appropriate file.

**kernel_address**

The memory address whereby the kernel image should always be loaded is called `kernel address`. By default, 32-bit kernels are loaded to address 0x8000, while 64-bit kernels are loaded to address 0x80000. Kernels are loaded to the address o xo if `kernel old` is specified.

**kernel_old**
To load the kernel to memory address oxo, set kernel old to 1.

**ramfsfile**

**Sramfsfile** is an optional filename for a ramfs to load on its boot partition.

**Note**

Multiple ramfs files can be loaded with newer firmware. You should use commas to separate the numerous file names, keeping in mind the 80-character line length limit. In memory, all of the loaded files are combined and processed as a single ramfs blob. On the boards, you can get more information.

**ramfsaddr**

The memory location where the ramfsfile should be loaded is ramfsaddr.

**initramfs**

Both ramfs filename and the memory address to load it are specified by the initramfs command. It combines the functionality of ramfsfile and ramfsaddr into a single argument. Followkernel (or o) can also be used to put it in
memory after the kernel image. *initramfs initramf.gz ox00800000* or *initramfs init.gz followkernel* are two examples of values.

Newer firmwares, like ramfsfile, allow you to load several files by separating their names with a comma.

**Note**

This option has a different syntax than the others, and you should avoid using the = character.

**init_uart_baud**

*init_uart_baud* is the initial baud rate of the UART. *115200* is the default value.

**init_uart_clock**

*init_uart_clock* is the initial clock frequency of the UART. *48000000* is the default value (48MHz).

It's worth noting that this clock only applies to UART0 (ttyAMA0 in Linux) and that the UART's maximum baudrate is 1/16th of the clock. UART1 (ttyS0 in Linux) is the default UART on the Pi 3 and Pi Zero, and its clock is the main VPU clock - at least 250MHz.
**bootcode_delay**

The bootcode delay command delays the start of loading in *bootcode.bin* for a specified number of seconds.

**Start elf: o** is the default value. This is very handy for adding a delay before reading the monitor's EDID, for example, if the Pi and display are powered from the same source but the monitor takes longer to boot up. If the display detection is incorrect on initial boot but correct when you soft-reboot the Pi without turning off the monitor, try adjusting this value.

**boot_delay**

The *boot_delay* command tells the computer to wait for a specific number of seconds in *start.elf* before loading the kernel: the default value is 1. The overall delay is calculated in milliseconds as \((1000 \times \text{boot}_\text{delay}) + \text{boot}_\text{delay}_\text{ms}\). This is important if your SD card takes a long time to prepare before Linux can boot from it.

**boot_delay_ms**
Together with $\text{boot delay}$, the $\text{boot delay ms}$ command implies wait for a specified amount of milliseconds in $\text{start.elf}$ before loading the kernel. 0 is the default value.

$\text{disable\_splash}$

The rainbow splash screen will not shown on boot if $\text{disable splash}$ is set to 1. 0 is the default value.

$\text{enable\_gic (Raspberry Pi 4 Only)}$

If this value is set to 0 on the Raspberry Pi 4B, interrupts will be directed to the ARM cores via the old interrupt controller rather than the GIC-400. 1 is the default value.

$\text{force\_eeprom\_read}$

To prohibit the firmware from attempting to read an I2C HAT EEPROM (attached to pins ID SD & ID SC) at powerup, set this option to 0. Also see disable poe fan.

$\text{os\_prefix}$

$\text{os prefix}$ is an alternative setting that lets you choose between different kernel versions and Device Tree files on the same card. Any value in $\text{os prefix}$ is appended to (put in front of) the name of any operating system files loaded
by the firmware, with "operating system files" defined as kernels, initramfs, cmdline.txt, .dtbs, and overlays.

The prefix is usually a directory name, but it could also be a part of the filename, like "test-." As a result, directory prefixes must include the / character at the end. In an attempt to reduce the chance of a non-bootable system, the firmware first tests the supplied prefix value for viability - unless the expected kernel and .dtb can be found at the new location/name, the prefix is ignored (set to "."). A special case of this viability test is applied to overlays, which will only be loaded from ${os_prefix}${overlay_prefix} (where the default value of overlay_prefix is "overlays") if ${os_prefix}${overlay_prefix}README exists, otherwise it ignores os_prefix and treats overlays as shared.

Here are two reasons why the firmware looks for key files rather than directories when checking prefixes: the prefix may not be a directory, and not all boot methods provide testing for the existence of a directory.)
Note

By employing an absolute path (with regard to the boot partition), any user-specified OS file can bypass all prefixes - simply start the file path with a /, e.g. `kernel=/my_common_kernel.img`.

See also `overlay_prefix` and `upstream_kernel`.

**overlay_prefix**

Specifies a subdirectory/prefix from which to load overlays - defaults to `overlays/` (note the trailing `/`). If used in conjunction with `os_prefix`, the `os_prefix` comes before the overlay_prefix, i.e. `dtoverlay=disable-bt` will attempt to load `${os_prefix}``${overlay_prefix}`disable-bt.dtbo.

Note

Unless `${os_prefix}``${overlay_prefix}`README exists, overlays are shared with the main OS (i.e. `os_prefix` is ignored).

**uart_2ndstage**

Setting `uart_2ndstage=1` causes the second-stage loader (`bootcode.bin` on devices prior to the Raspberry Pi 4, or the boot code in the EEPROM for Raspberry Pi 4 devices)
and the main firmware (\texttt{start*.elf}) to output diagnostic information to UART0.

Be warned that unless output is disabled (\texttt{dtoverlay=disable-bt}) or switched to the other UART (\texttt{dtoverlay=miniuart-bt}), output will likely interfere with Bluetooth functionality, and if the UART is accessed simultaneously with output from Linux, data loss may occur, resulting in distorted output.

Only use this function if you're attempting to figure out why your computer isn't booting up.

\textbf{upstream\_kernel}

If \texttt{upstream kernel=1} is used, the firmware sets \texttt{os prefix} to "upstream/" unless it is expressly set to something different, but it will be disregarded if the requisite kernel and.dtb file cannot be located while utilizing the prefix, much like other \texttt{os prefix} values.

For DTBs, the firmware will also prefer upstream Linux names (\texttt{bcm2837-rpi-3-b.dtb} instead of \texttt{bcm2710-rpi-3-b.dtb}, for example).
If the upstream file cannot be retrieved, the firmware will load the downstream variant and apply the "upstream" overlay to make certain changes.

It's worth noting that this happens after the *os prefix* has been set.

**GPIO Control**

**gpio**

The *gpio* directive allows GPIO pins to be set to specific modes and values at boot time in a way that would previously have needed a custom *dt-blob.bin* file. Each line applies the same settings (or at least makes the same changes) to a set of pins, either a single pin (3), a range of pins (3-4), or a comma-separated list of either (3-4,6,8). The pin set is followed by an = and one or more comma-separated attributes from this list:

- *ip* - Input
- *op* - Output
- *ao-a5* - Alto-Alt5
- *dh* - Driving high (for outputs)
- *dl* - Driving low (for outputs)
- *pu* - Pull up
- **pd** - Pull down
- **pn/np** - No pull

**gpio** Setting are applied in order, so those that appear later take precedence over those that appear earlier.

Examples:

```bash
# Select Alt2 for GPIO pins 0 to 27 (for DPI24)
gpio=0-27=a2

# Set GPIO12 to be an output set to 1
gpio=12=op,dh

# Change the pull on (input) pins 18 and 20
gpio=18,20=pu

# Make pins 17 to 21 inputs
gpio=17-21=ip
```

Because the **gpio** directive respects the "[...]
 section headers in **config.txt**, multiple settings can be used depending on the model, serial number, and EDID.

This technique has no direct influence on the kernel; it does not cause GPIO pins to be exported to the sysfs
interface, and it can be overridden by pinctrl entries in the Device Tree as well as programs like raspi-gpio.

It's also worth noting that there's a few seconds of delay between turning on the computer and the adjustments taking effect - longer if you're booting over the network or from a USB mass storage device.

**enable_jtag_gpio**

Enabling the JTAG interface for the ARM CPU by setting `enable_jtag_gpio=1` enables Alt4 mode for GPIO pins 22-27 and sets up some internal SoC connections.

It is compatible with all Raspberry Pi models.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO22</td>
<td>ARM_TRST</td>
</tr>
<tr>
<td>GPIO23</td>
<td>ARM_RTCK</td>
</tr>
<tr>
<td>GPIO24</td>
<td>ARM_TDO</td>
</tr>
<tr>
<td>GPIO25</td>
<td>ARM_TCK</td>
</tr>
<tr>
<td>GPIO26</td>
<td>ARM_TDI</td>
</tr>
<tr>
<td>GPIO27</td>
<td>ARM_TMS</td>
</tr>
</tbody>
</table>

**Overclocking Options**
The kernel has a CPUFreq driver with the "powersave" governor enabled by default, switched to "ondemand" during boot, when raspi-config is installed. With "ondemand" governor, CPU frequency will vary with processor load. You can change the minimum values using the *_min config options, or you can deactivate dynamic clocking using a static scaling governor ("powersave" or "performance") or force turbo=1.

Overclocking and overvoltage will be disabled at runtime when the SoC reaches temp_limit (see below), which defaults to 85°C, in order to cool down the SoC. You should not hit this limit with Raspberry Pi Models 1 and 2, but you are more likely to with Raspberry Pi 3 and Raspberry Pi 4B. When an undervoltage scenario is recognized, overclocking and overvoltage are likewise disabled.

Note

See the frequency management and heat control sections for more details.

Warning

Setting any overclocking parameters other than those used by raspi-config may cause a permanent bit to be set within
the SoC, allowing you to determine whether your Raspberry Pi has been overclocked.

If `force turbo` is set to 1 and any of the `over voltage_*` options is set to a value greater than 0, the overclock bit will be set. For more information, see the Turbo Mode blog page.

**Overclocking**
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm_freq</td>
<td>Frequency of the ARM CPU in MHz.</td>
</tr>
<tr>
<td>gpu_freq</td>
<td>Sets core_freq, h264_freq, isp_freq, v3d_freq and hevc_freq together.</td>
</tr>
<tr>
<td>core_freq</td>
<td>Frequency of the GPU processor core in MHz; influences CPU performance because it drives the L2 cache and memory bus; the L2 cache benefits only Pi Zero/Pi Zero W/ Pi 1, there is a small benefit for SDRAM on Pi 2/Pi 3. See section below for use on the Pi 4.</td>
</tr>
<tr>
<td>h264_freq</td>
<td>Frequency of the hardware video block in MHz; individual override of the gpu_freq setting</td>
</tr>
<tr>
<td>isp_freq</td>
<td>Frequency of the image sensor pipeline block in MHz; individual override of the gpu_freq setting</td>
</tr>
<tr>
<td>v3d_freq</td>
<td>Frequency of the 3D block in MHz; individual override of the gpu_freq setting</td>
</tr>
<tr>
<td>hevc_freq</td>
<td>Frequency of the High Efficiency Video Codec block in MHz; individual override of the cpu_freq setting; Pi 4 only.</td>
</tr>
<tr>
<td>sdram_freq</td>
<td>Frequency of the SDRAM in MHz; SDRAM overlocking on Pi 4B is not currently supported</td>
</tr>
<tr>
<td>over_voltage</td>
<td>CPU/GPU core upper voltage limit. The value should be in the range [-16,8] which equates to the range [0.95V,1.55V] (0.8V on RPi 1) with 0.025V steps. In other words, specifying -16 will give 0.95V (0.8V on RPi 1) as the maximum CPU/GPU core voltage, and specifying 8 will allow up to 1.55V (1.4V on RPi 1). For defaults see table below. Values above 8 are only allowed when force_turbine is specified; this sets the warranty bit if over_voltage &gt; 0 is also set.</td>
</tr>
<tr>
<td>over_voltage_sdram</td>
<td>Sets over_voltage_sdram_a, over_voltage_sdram_d and over_voltage_sdram_p together.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>over_voltage_sdram_c</td>
<td>SDRAM controller voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps.</td>
</tr>
<tr>
<td>over_voltage_sdram_i</td>
<td>SDRAM I/O voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps.</td>
</tr>
<tr>
<td>over_voltage_sdram_p</td>
<td>SDRAM phy voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps.</td>
</tr>
<tr>
<td>force_turbo</td>
<td>Forces turbo mode frequencies even when the ARM cores are not busy. Enabling this may set the warranty bit if <code>over_voltage</code> is also set.</td>
</tr>
<tr>
<td>initia_turbo</td>
<td>Enables turbo mode from boot for the given value in seconds, or until cpufreq sets a frequency. The maximum value is 80.</td>
</tr>
<tr>
<td>arm_freq_min</td>
<td>Minimum value of <code>arm_freq</code> used for dynamic frequency clocking. Note that reducing this value below the default does not result in any significant power savings and is not currently supported.</td>
</tr>
<tr>
<td>core_freq_min</td>
<td>Minimum value of <code>core_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>gpu_freq_min</td>
<td>Minimum value of <code>gpu_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>h264_freq_min</td>
<td>Minimum value of <code>h264_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>isp_freq_min</td>
<td>Minimum value of <code>isp_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>v3d_freq_min</td>
<td>Minimum value of <code>v3d_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>hevc_freq_min</td>
<td>Minimum value of <code>hevc_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>sdram_freq_min</td>
<td>Minimum value of <code>sdram_freq</code> used for dynamic frequency clocking.</td>
</tr>
<tr>
<td>over_voltage_min</td>
<td>Minimum value of <code>over_voltage</code> used for dynamic frequency clocking. The value should be in the range [-16,8] which equates to the range [0.8V,1.4V] with 0.025V steps. In other words, specifying -16 will give 0.8V as the CPU/GPU core idle voltage, and specifying 8 will give a minimum of 1.4V.</td>
</tr>
<tr>
<td>temp_limit</td>
<td>Overheat protection. This sets the clocks and voltages to default when the SoC reaches this value in degree Celsius. Values over 85 are clamped to 85.</td>
</tr>
<tr>
<td>temp_soft_limit</td>
<td><strong>3A+/3B+ only</strong> CPU speed throttle control. This sets the temperature at which the CPU clock speed throttling system activates. At this temperature, the clock speed is reduced from 1400MHz to 1200MHz. Defaults to 60, can be raised to a maximum of 90, but this may cause instability.</td>
</tr>
</tbody>
</table>
The default values for the choices on various Raspberry Pi models are listed in this table; all frequencies are in MHz.

<table>
<thead>
<tr>
<th>Option</th>
<th>Pi 0/W</th>
<th>Pi 1</th>
<th>Pi 2</th>
<th>Pi 3</th>
<th>Pi3A+/Pi3B+</th>
<th>Pi4/C/M4</th>
<th>Pi 400</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm_freq</td>
<td>1000</td>
<td>700</td>
<td>900</td>
<td>1200</td>
<td>1400</td>
<td>1500</td>
<td>1800</td>
</tr>
<tr>
<td>core_freq</td>
<td>400</td>
<td>250</td>
<td>250</td>
<td>400</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>h264_freq</td>
<td>300</td>
<td>250</td>
<td>250</td>
<td>400</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>isp_freq</td>
<td>300</td>
<td>250</td>
<td>250</td>
<td>400</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>v3d_freq</td>
<td>300</td>
<td>250</td>
<td>250</td>
<td>400</td>
<td>400</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>hevc_freq</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>sdram_freq</td>
<td>450</td>
<td>400</td>
<td>450</td>
<td>450</td>
<td>500</td>
<td>3200</td>
<td>3200</td>
</tr>
<tr>
<td>arm_freq_min</td>
<td>700</td>
<td>700</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>core_freq_min</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>gpu_freq_min</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>h264_freq_min</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>isp_freq_min</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>v3d_freq_min</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>sdram_freq_min</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>3200</td>
<td>3200</td>
</tr>
</tbody>
</table>

This table lists the defaults for parameters that apply to all models.
The firmware uses Adaptive Voltage Scaling (AVS) to determine the optimum CPU/GPU core voltage in the range defined by `over_voltage` and `over_voltage_min`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial_turbo</td>
<td>0 (seconds)</td>
</tr>
<tr>
<td>temp_limit</td>
<td>85 (°C)</td>
</tr>
<tr>
<td>over_voltage</td>
<td>0 (1.35V, 1.2V on RPi 1)</td>
</tr>
<tr>
<td>over_voltage_min</td>
<td>0 (1.2V)</td>
</tr>
<tr>
<td>over_voltage_sdram</td>
<td>0 (1.2V)</td>
</tr>
<tr>
<td>over_voltage_sdram_c</td>
<td>0 (1.2V)</td>
</tr>
<tr>
<td>over_voltage_sdram_i</td>
<td>0 (1.2V)</td>
</tr>
<tr>
<td>over_voltage_sdram_p</td>
<td>0 (1.2V)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Default</th>
<th>Resulting voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi 1</td>
<td>0</td>
<td>1.2V</td>
</tr>
<tr>
<td>Pi 2</td>
<td>0</td>
<td>1.2-1.3125V</td>
</tr>
<tr>
<td>Pi 3</td>
<td>0</td>
<td>1.2-1.3125V</td>
</tr>
<tr>
<td>Pi 4, Pi400, CM4</td>
<td>0</td>
<td>0.88V</td>
</tr>
<tr>
<td>Pi Zero</td>
<td>6</td>
<td>1.35V</td>
</tr>
</tbody>
</table>
Conditional Filters

When using a single SD Card (or card image) with one Raspberry Pi and one monitor, it's simple to configure `config.txt` for that exact combination and leave it alone, only changing it when something changes.

A single set of settings may not be sufficient if one Raspberry Pi is swapped between various monitors or the SD card (or card image) is swapped between numerous boards. Conditional filters allow you to specify which sections of the config file should be utilized only in specified circumstances, allowing a single `config.txt` to generate various configurations depending on the hardware.

The [all] filter

The most basic filter is the [all] filter. It clears any previously configured filters and allows you to apply any of the settings specified below to all hardware. To avoid mistakenly merging filters, it's usually a good idea to include a [all] filter at the conclusion of groups of filtered settings (see below).

Model Filters
The conditional model filters are applied as shown in the table below.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Applicable model(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pi1]</td>
<td>Model A, Model B, Compute Module</td>
</tr>
<tr>
<td>[pi2]</td>
<td>Model 2B (BCM2836- or BCM2837-based)</td>
</tr>
<tr>
<td>[pi3]</td>
<td>Model 3B, Model 3B+, Model 3A+, Compute Module 3, Compute Module 3+</td>
</tr>
<tr>
<td>[pi3+]</td>
<td>Model 3A+, Model 3B+</td>
</tr>
<tr>
<td>[pi4]</td>
<td>Model 4B, Pi 400, Compute Module 4</td>
</tr>
<tr>
<td>[pi400]</td>
<td>Pi 400</td>
</tr>
<tr>
<td>[cm4]</td>
<td>Compute Module 4</td>
</tr>
<tr>
<td>[p10w]</td>
<td>Zero W, Zero WH</td>
</tr>
<tr>
<td>[p102]</td>
<td>Zero 2 W</td>
</tr>
<tr>
<td>[board-type=Type]</td>
<td>Filter by type number - see Raspberry Pi Revision Codes E.g. [board-type=0101] would match CM4.</td>
</tr>
</tbody>
</table>

Because the Pi 1 and Pi 2 require distinct kernels, these are especially handy for defining different kernel, initramfs, and cmdline settings. Because the Pi 1 and Pi 2 have different default speeds, they can also be used to specify various overclocking settings.

To give an example, to define distinct initramfs images for each:

[dir]

initramfs initrd.img-3.18.7+ followkernel

[p2]


\texttt{initramfs initrd.img-3.18.7-v7+ followkernel}

\texttt{[all]}

Ensure to use the \texttt{[all]} filter at the end so that any following settings aren't restricted to the Pi 2 hardware. It's vital to notice that \texttt{[piow]} AND \texttt{[pio]} will be visible to the Raspberry Pi Zero W.

The \texttt{[none]} filter

The \texttt{[none]} filter prohibits any settings from being applied to any device in the following steps. Although you can't do anything without \texttt{[none]}, it's a convenient method to preserve groupings of useless options in config.txt without needing to comment out every line.

The \texttt{[EDID=*]} filter

This allows specific settings to be picked based on the monitors' EDID identifiers when moving between several monitors when using a single SD card in your Pi and where a blank config isn't sufficient to automatically determine the desired resolution for each one.
Run the following command to see the EDID name of an attached monitor:

`tvservice -n`

This will result in a printout that looks like this:

`device_name=VSC-TD2220`

Then you can provide parameters that are specific to this monitor:

`[EDID=VSC-TD2220]`

`hdmi_group=2`

`hdmi_mode=82`

`[all]`

This forces the specified monitor into 1920x1080 DVT mode, without impacting any other monitors. Because these settings only apply at boot, the monitor must be attached and the Pi must be able to read its EDID information to determine the right name. After booting, hotplugging a different monitor into the Pi will not change the settings.
If both HDMI ports on the Raspberry Pi 4 are in use, the EDID will be checked against both of them, and subsequent setup will only be applied to the first matched device.

To get the EDID names for both ports, use `tvservice -l` to list all attached devices in a terminal window, then use the returned numerical IDs in `tvservice -v id> -n` to find the EDID name for a specific display ID. **The Serial Number Filter**

Even if you replace the SD card to a different one, some settings should be applied to a single unique Pi.

License keys and overclocking settings are two examples (although the licence keys already support SD card swapping in a different way).

Even if EDID identification is not possible, you can use this to select alternative display settings, as long as you don't swap displays between your Pis. If your monitor doesn't have a suitable EDID name, or if you're using composite output, for example (for which EDID cannot be read).
Enter the following command to see your Pi's serial number:

```
cat /proc/cpuinfo
```

At the bottom, the serial will be displayed as a 16-digit hex value.

For instance, suppose you see the following:

```
Serial          : 0000000012345678
```

Then you can specify parameters that will be applied just to this Pi:

```
[0x12345678]
# settings here are applied only to the Pi with this serial

[all]
# settings here are applied to all hardware
```

The GPIO Filter

Filtering can also be done based on the state of a GPIO.

As an example,
[gpio4=1]

# Settings here are applied if GPIO 4 is high

[gpio2=0]

# Settings here are applied if GPIO 2 is low

[all]

# settings here are applied to all hardware

The [HDMI:*] Filter

Note

This filter is only compatible with the Raspberry Pi 4. Because the Raspberry Pi 4 has two HDMI ports, many config.txt commands involving HDMI need specifying which HDMI port is being referenced to. Following HDMI setups are filtered to the specified port by the HDMI conditional.

[HDMI:0]

hdmi_group=2

hdmi_mode=45

[HDMI:1]
On all port-specific HDMI instructions, an alternate `variable:index` syntax is available. You might use the following example, which is similar to the previous one:

```
hdmi_group:0=2
hdmi_mode:0=45
hdmi_group:1=2
hdmi_mode:1=67
```

**Combining Conditional Filters**

Because it is impossible for two filters of the same type to be true at the same time, `[pi2] overrides [pi1]`.

Filters of various types can be coupled by simply listing them one after the other, as in:

```
# settings here are applied to all hardware
[EDID=VSC-TD2220]
# settings here are applied only if monitor VSC-TD2220 is connected
```
[pi2]

# settings here are applied only if monitor VSC-TD2220 is connected *and* on a Pi 2

[all]

# settings here are applied to all hardware

To avoid accidentally merging different filter types, use the [all] filter to reset all prior filters.

Miscellaneous Options

avoid_warnings

This option can be used to disable the warning symbols, although it is not recommended.

avoid_warnings=1  The warning overlays are turned off.
avoid_warnings=2  The warning overlays are disabled, however turbo mode is enabled even when low voltage is present.

logging_level

Sets the logging level for VideoCore.

The value is a bitmask particular to VideoCore.
include

This command inserts the contents of the supplied file into the current file.

For example, adding the line `include extraconfig.txt` to `config.txt` will include the content of `extraconfig.txt` file in the `config.txt` file.

Include directives are not supported by bootcode.bin or the EEPROM bootloader

max_usb_current

Warning

This command has been deprecated and no longer works. The USB ports on some Raspberry Pi models were originally limited to 600mA. This default was modified to 1200mA by setting `max usb current=1`. This option is no longer necessary because all firmware now has this flag set by default.

disable_poe_fan

Set this option to 1 to disable I2C control of the PoE HAT fan (on ID SD & ID SC pins).
Even if the PoE HAT is not used, a probe on the I2C bus will occur at startup if this is not done.

<table>
<thead>
<tr>
<th>Ratio</th>
<th>aspect_ratio ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:3</td>
<td>1</td>
</tr>
<tr>
<td>14:9</td>
<td>2</td>
</tr>
<tr>
<td>16:9</td>
<td>3</td>
</tr>
<tr>
<td>5:4</td>
<td>4</td>
</tr>
<tr>
<td>16:10</td>
<td>5</td>
</tr>
<tr>
<td>15:9</td>
<td>6</td>
</tr>
<tr>
<td>21:9</td>
<td>7</td>
</tr>
<tr>
<td>64:27</td>
<td>8</td>
</tr>
</tbody>
</table>

To specify the HDMI port on the Pi4, add an index identifier to the config.txt file, for example, `hdmi cvt:o=....hdmi timings:1=....`.

When no port identifier is specified, the settings are applied to port 0.
How to troubleshoot an HDMI cable

In rare cases, you may need to increase the HDMI drive strength, for example, if your display has speckling or you're using extremely long cables. This can be accomplished by including the `config hdmi boost config.txt` item on the config.txt video page.

Nota bene, configuration HDMI boost is not yet supported on the Raspberry Pi 4B; this feature will be added in a future software update.

Rotation of the Monitor

The options for rotating the display on your Raspberry Pi are determined by the installed display driver software, which is also determined by the Raspberry Pi model. This is the default configuration for the Raspberry Pi 4 Model B. If you're using the Raspberry Pi desktop, the Screen Configuration Utility allows you to rotate the screen.

Preferences menu on the desktop. This will bring up a graphical representation of the Raspberry Pi's display or displays. Select the required option by right clicking on the display you wish to rotate.
Additionally, these settings can be changed via the command line `xrandr` option. The following commands perform rotations of $0^\circ$, $-90^\circ$, $+90^\circ$, and $180^\circ$, respectively.

```
xrandr --output HDMI-1 --rotate normal
xrandr --output HDMI-1 --rotate left
xrandr --output HDMI-1 --rotate right
xrandr --output HDMI-1 --rotate inverted
```

Take note that the `--output` option specifies the device to which the rotation should be applied. You can determine the device name by typing `xrandr` on the command line, which will display information about all attached devices, including their names.

Additionally, you can use the `--reflect` option on the command line to mirror the display. The reflection can be one of the 'standard' 'x', 'y', or 'xy' types. This results in a reflection of the output contents along the specified axes. For instance:

```
xrandr --output HDMI-1 --reflect
```

If you are only using the console (no graphical desktop), you will need to configure the kernel command line flags.
appropriately. Modify the console settings according to the instructions on the page.

**Driver for Legacy Graphics**

**Note**

This is the default configuration for all Raspberry Pi models prior to the Raspberry Pi 4 Model B. When using legacy display drivers, there are config.txt options for rotating.

The display hdmi rotate method rotates the HDMI display, whereas display lcd rotate rotates any attached LCD panel (using the DSI or DPI interface). Both the desktop and console can be rotated using these options. Each of the following options requires one of the following parameters:

<table>
<thead>
<tr>
<th>display_*_rotate</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no rotation</td>
</tr>
<tr>
<td>1</td>
<td>rotate 90 degrees clockwise</td>
</tr>
<tr>
<td>2</td>
<td>rotate 180 degrees clockwise</td>
</tr>
<tr>
<td>3</td>
<td>rotate 270 degrees clockwise</td>
</tr>
<tr>
<td>0x10000</td>
<td>horizontal flip</td>
</tr>
<tr>
<td>0x20000</td>
<td>vertical flip</td>
</tr>
</tbody>
</table>
Please keep in mind that the 90 and 270 degree rotation options require additional GPU memory, and thus will not work with the 16MB GPU split.

By adding the rotation and flip settings together, you can combine them. Additionally, you can perform horizontal and vertical flips in the same manner. For example, \(0x20000 + 0x10000 + 2 = 0x30002\).

**Configuration Audio**

The Raspberry Pi supports up to three audio output modes: HDMI 1 and 2, and a headphone jack, if they are present. At any time, you can switch between these modes.

If your HDMI monitor or TV includes built-in speakers, audio can be played through the HDMI cable, or via headphones or other speakers connected via the headphone jack. If your display claims to have speakers, audio is typically output via HDMI; if not, it is output via the headphone jack. This may not be the desired output configuration, or the auto-detection may be inaccurate; in either case, you can switch the output manually.
Changing the Output of the Audio

The audio output can be configured in two ways: via the Desktop volume control or via the raspi-config command line tool.

Utilization of the Desktop

By right-clicking the volume icon in the desktop taskbar, you can access the audio output selector, which enables you to switch between internal audio outputs. Additionally, you can choose from a variety of external audio devices, including USB sound cards and Bluetooth audio devices. The currently selected audio output device is indicated by a green tick — simply left-click the desired output in the pop-up menu.

This can be changed via the menu. Volume control and mute are device-specific.

Utilization of raspi-config

To launch raspi-config, type the following command into the command line:

`sudo raspi-config`
This will bring up the configuration screen as follows:

- Press Enter to select System Options (currently option 1, but yours may be different).
- Now, select the Audio option (currently S2, but yours may vary) and press Enter:
- Select the desired mode, press Enter, and then the right arrow key to close the options list. Finally, select Finish to close the configuration tool.
- After making changes to your audio settings, you must restart your Raspberry Pi for the changes to take effect.

**How to troubleshoot your HDMI**

Occasionally, it may be necessary to edit config.txt to force HDMI mode (as opposed to DVI mode, which does not send sound). This can be accomplished by editing /boot/config.txt and setting hdmi drive=2, followed by a reboot.

**Configuration of External Storage**

You can connect an external hard disk, solid state drive, or USB stick to any of the Raspberry Pi's USB ports and mount the file system to gain access to the data stored on it.
By default, the Raspberry Pi mounts several popular file systems, including FAT, NTFS, and HFS+, in the `/media/pi/HARD-DRIVE-LABEL> directory.

Nota bene, Raspberry Pi OS Lite omits automounting.

To configure your storage device so that it always mounts to a specified location, you must manually mount it.

**How to Mount a Storage Device**

You can choose to mount your storage device in a specific folder. Typically, this is done within the `/mnt` folder, for example, `/mnt/mydisk`. Please keep in mind that the folder must be empty.

1. Connect the storage device to the Raspberry Pi’s USB port.

2. Using the following command, list all the disk partitions on the Raspberry Pi:

```
sudo lsblk -o UUID,NAME,FSTYPE,SIZE,MOUNTPOINT,LABEL,MODEL
```
The Raspberry Pi makes use of the filesystem mount points / and /boot. This list will include your storage device, as well as any other connected storage.

3. Using the SIZE, LABEL, and MODEL columns, determine the disk partition's name that points to your storage device. For instance, sda1.

4. The FSTYPE column contains information about the filesystem. Install the exFAT driver if your storage device uses the exFAT file system:

   apt-get update sudo

   apt install exfat-fuse sudo

5. If your storage device is formatted with the NTFS file system, you will have read-only access. To write to the device, you can install the ntfs-3g driver as follows:

   apt-get update sudo

   apt install ntfs-3g sudo

6. Execute the following command to determine the disk partition's location:

   blkid sudo
For instance, /dev/sda1.

7. Create a target folder that will serve as the storage device's mount point. In this case, the mount point is called mydisk. You can specify a different name with sudo mkdir /mnt/mydisk.

8. Mount the storage device to the newly created mount point:

   ```
   mount /dev/sda1 /mnt/mydisk sudo
   ```

9. Verify the storage device's successful mounting by listing its contents.

   ```
   ls /mnt/mydisk /mnt/mydisk
   ```

Configuring Automatic Mounting

You can edit the fstab file to specify the directory in which the storage device will be mounted automatically when the Raspberry Pi boots. The fstab file specifies the universally unique identifier for the disk partition (UUID).

1. Obtain the disk partition's UUID.

   ```
   blkid sudo
   ```
2. From the list, locate the disk partition and note the UUID. As an illustration, consider 5C24-1453.

3. Using a command-line editor such as nano, open the fstab file:

/etc/fstab nano sudo

4. In the fstab file, add the following line:

/mnt/mydisk fstype defaults,auto,users,rw,nofail
UUID=5C24-1453 0 0

Replace file system with the file system type that was discovered in step 2 of 'Mounting a storage device' above, for example: ntfs.

5. If the file system is FAT or NTFS, immediately after nofail, add, mask=000 - this will grant all users read/write access to all files on the storage device.

After creating an entry in fstab, you can boot your Raspberry Pi with or without the attached storage device. Prior to unplugging the device, either power down the Pi or manually unmount it using the steps in

Below is a section titled 'Unmournung a storage device.'
If the storage device is not connected when the Pi starts, it will take an additional 90 seconds to boot. This can be shortened by immediately following `nofail with,x-system. Device-timeout=30` in step 4. This sets the timeout to 30 seconds, which means the system will attempt to mount the disk for no more than 30 seconds.

For more information on each Linux command, use the main command to navigate to the appropriate manual page. For instance, `man fstabs`.

**Mounting and Unmounting a Storage Device**

When the Raspberry Pi is powered down, the system automatically unmounts the storage device, making it safe to unplug. To unmount a device manually, use the following command:

```
/mnt/mydisk umount sudo
```

If you receive an error indicating that the 'target is busy,' this indicates that the storage device was not unmounted properly. If no error message was displayed, you can safely unplug the device at this point.

Dealing with the 'target is busy' condition
The message 'target is busy' indicates that a program is utilizing files on the storage device. Use the following procedure to close the files.

1. Exit any program that is currently open on the storage device.

2. If you are using a terminal, ensure that you are not in the folder where the storage device is mounted, or a sub-folder thereof.

3. If you are still unable to unmount the storage device, you can use the lsof command to determine which programs are currently accessing the device's files. To begin, you must install lsof via apt:

   
   **sudo apt update**

   **sudo apt install lsof**

To use lsof:

   **lsof /mnt/mydisk**
Localising your Raspberry Pi
You can set your Raspberry Pi up to match your regional settings.

Changing the Language
If you want to select a different language use raspi-config.

Configuring the Keyboard
If you want to select a different keyboard use raspi-config.

Changing the Timezone
Once again, this is something you can change using the raspi-config tool.

Changing the default pin configuration
Edit this on GitHub

Warning
This feature is intended for advanced users.

As of July 2014, the Raspberry Pi firmware supports custom default pin configurations through a user-provided Device Tree blob file. To find out whether your firmware is recent enough, please run vcgencmd version.
Device Pins During Boot Sequence

During the bootup sequence, the GPIO pins go through various actions.

1 Power-on — pins default to inputs with default pulls; the default pulls for each pin are described in the datasheet

2 Setting by the bootrom

3 Setting by bootcode.bin

5 GPIO command in config.txt

Added six firmware pins (e.g. UARTS)

7 Tree of Kernels/Device

The same procedure applies during a soft reset, except for default pulls, which are applied only during a power-on reset.

Take note that the transition from stage 1 to stage 4 may take a few seconds. GPIO pins may not be in the state expected by attached peripherals during this time period (as defined in dtblob.bin or config.txt). Due to the fact that different GPIO pins have varying default pull values, you should implement one of the following for your peripheral:
1 Select a GPIO pin that defaults to pulls on reset, as required by the peripheral.

2 Defer startup of the peripheral until stage 4/5 is reached.

3 Include a suitable pull-up/pull-down resistor.

Creating a Personalized Device Tree Blob

To compile a Device Tree source file (.dts),

4 Setting by dt-blob.bin (this page) (this page)

5 Setting by the GPIO command in config.txt

6 Additional firmware pins (e.g. UARTS) (e.g. UARTS)

7 Kernel/Device Tree

On a soft reset, the same procedure applies, except for default pulls, which are only applied on a power-on reset.

Note that it may take a few seconds to get from stage 1 to stage 4. During that time, the GPIO pins may not be in the state expected by attached peripherals (as defined in dtblob.bin or config.txt) (as defined in dtblob.bin or config.txt). Since different GPIO pins have different default pulls, you should do one of the following for your peripheral:
1 Choose a GPIO pins that defaults to pulls as required by the peripheral on reset

2 Delay the peripheral’s startup until stage 4/5 has been reached

3 Add an appropriate pull-up/-down resistor

**Providing a Custom Device Tree Blob**

In order to compile a Device Tree source (.dts) file into a Device Tree blob (.dtb) file, the Device Tree compiler must be installed by running `sudo apt install device-tree-compiler`. The dtc command can then be used as follows:

```
sudo dtc -I dts -O dtb -o /boot/dt-blob.bin dt-blob.dts
```

Similarly, a .dtb file can be converted back to a .dts file, if required.

```
dtc -I dtb -O dts -o dt-blob.dts /boot/dt-blob.bin
```

sections of the dt-blob

The dt-blob.bin is used to configure the binary blob (VideoCore) at boot time

. It is not currently used by the Linux kernel, but when we reconfigure the Raspberry Pi kernel to use a dt-blob for configuration, a kernel section will be added. The dt-blob
can be used to configure all Raspberry Pi versions, including the Compute Module, to use the alternate settings. In the dt-blob, the following sections are valid:

1. **Video core**

This section contains all of the information about the Video Core blob. This section must contain all subsequent sections.

2. **Pins**

There are several distinct pins_* sections, each devoted to a specific Raspberry Pi model, namely:

- Pins rev1 Rev 1 pin configuration. There are some differences due to the relocation of the I2C pins.
  - Pins rev2 Rev2 pin configuration. This includes the additional pins on P5 for codecs.
  - Pins bplus1 Model B+ revision 1.1, complete with 40-pin connector.
  - Pins bplus2 Model B+ revision 1.2, with the low-power and lan-run pins switched.
- Pins aplus A+ model without Ethernet.
- pins_aplus Model A+, lacking Ethernet.
- pins_2b1 Pi 2 Model B rev 1.0; controls the SMPS via I2C.
- pins_2b2 Pi 2 Model B rev 1.1; controls the SMPS via software I2C on 42 and 43.
- pins_3b1 Pi 3 Model B rev 1.0
- pins_3b2 Pi 3 Model B rev 1.2
- pins_3bplus Pi 3 Model B+
- pins_3aplus Pi 3 Model A+
- pins_pio The Pi Zero
- pins_piow The Pi Zero W
- pins_cm The Compute Module. The default for this is the default for the chip, so it is a useful source of information about default pull ups/downs on the chip.
- pins_cm3 The Compute Module version 3
- Each pins_* section can contain pin_config and pin_defines sections
**pin_config**

Individual pins are configured in the pin config section. Each item in this section must be a named pin section, such as pin@p32, which stands for General Purpose Input/Output. There is a special section called pin@default that contains the default settings for anything that is not explicitly named in the pin config section.

**pin@pinname**

This section can contain any combination of the following items:

- polarity
  - active_high
  - active_low
- termination
  - pull_up
pull_down

no_pulling

startup_state

active

inactive

function

input

output

sdcard

i2co

i2c1

spi2

spi

mi

169
Drive strength mA The drive strength is used to establish the pins’ strength. Please keep in mind that the bank can only have a single drive strength specified. The values 8> and 16> are valid.

pin defines

This section is used to associate particular VideoCore functionality with specific pins. This enables the user to move the camera power enable pin to somewhere different, or move the HDMI hotplug position: things that Linux
does not control. Please see the sample DTS file provided below.

**Configuration of the Clock**

It is possible to modify the configuration of the clocks via this interface, though the results may be unpredictable! The clocking system's configuration is quite complicated. Each of the five PLLs has its own fixed (or variable, in the case of the PLLC) VCO frequency. Each VCO then has a number of distinct channels that can be configured using various divisions of the VCO frequency. Each of the clock destinations can be configured to come from one of the clock channels, although there is a restricted mapping of source to destination, so not all channels can be routed to all clock destinations.

Here are a few sample configurations for modifying specific clocks. This resource will be updated as requests for clock configurations are received.
The above will set the PLLA to a source VCO running at 1.96608GHz (the limits for this VCO are 600MHz - 2.4GHz), change the APER channel to /4, and configure GPCLK0 to be sourced from PLLA through APER. This is used to give an audio codec the 12288000Hz it needs to produce the 48000 range of frequencies.

Sample Device Tree Source File

The example file comes from the firmware repository, https://github.com/raspberrypi/firmware/blob/master/extra/dt-blob.dts. This is the master Raspberry Pi blob from which all subsequent blobs are typically derived.

Device Trees, Overlays, and Parameters

Raspberry Pi kernels and firmware use a Device Tree (DT) to describe the hardware present in the Pi. These Device
Trees may include DT parameters that provide a degree of control over some onboard features. DT overlays allow optional external hardware to be described and configured, and they also support parameters for more control.

The firmware loader (start. Elf and its variants) is responsible for loading the DTB (Device Tree Blob - a machine readable DT file) (Device Tree Blob - a machine readable DT file). It determines which one to load based on the revision number of the board and makes some modifications to further customize it (memory size, Ethernet addresses etc.). This runtime customization avoids the need for lots of DTBs with only minor differences.

Config.txt is scanned for user-provided parameters, along with any overlays and their parameters, which are then applied. The loader examines the result to learn (for example) which UART, if any, is to be used for the console. Finally, it starts the kernel, passing a pointer to the merged DTB as an argument.
Trees of Devices

A Device Tree (DT) is a representation of a system's hardware. It should contain the name of the base CPU, the configuration of its memory, and any peripherals (internal and external). A DT should not be used to describe the software, though listing the hardware modules usually results in the loading of driver modules. It helps to keep in mind that DTs are supposed to be OS-neutral, which means that anything Linux-specific should probably be excluded.

A Device Tree is a hierarchical representation of the hardware configuration. Each node may contain subnodes and properties. Properties are named arrays of bytes that may contain strings, big-endian numbers, arbitrary sequences of bytes, or any combination thereof. Nodes are analogous to directories and properties to files in a filesystem. A path can be used to describe the locations of nodes and properties within the tree, with slashes acting as separators and a single slash (/) indicating the root.
Syntax fundamentals for DTS

Device Trees are typically written in a text format called Device Tree Source (DTS) and stored in files ending in .dts. The syntax of DTS is similar to that of C, with braces used for grouping and semicolons at the end of each line. Take note that DTS is mandatory.

After closing braces, use semicolons to represent C struts rather than functions. The compiled binary format is called Flattened Device Tree (FDT) or Device Tree Blob (DTB), and it is saved in .dtb files.

A simple tree in the .dts format is as follows:
This tree contains the following:

• A mandatory header: /dts-v1/.

• The inclusion of another DTS file, conventionally named *.dtsi and analogous to a C's.h header file - see below for an aside about /include/.

• A solitary root node:

• A pair of child nodes named node1 and node2

Node1 has two children: child-node1 and child-node2.

• a label (cousin) and a reference to that label (&cousin): see the section below on Labels and References.

Numerous properties dispersed throughout the tree

• a repeated node (/node2) - see below for an aside on /include/.

Properties are straightforward key-value pairs in which the value can be null or an arbitrary byte stream. While data types are not encoded in the data structure, a Device Tree
source file can express a few fundamental data representations.

Double quotes are used to denote text strings (NUL-terminated):

```javascript
var string-property = "a string"; var string-property = "a string";
```

Cells are unsigned 32-bit integers that are delimited by angle brackets:

```javascript
0xbeef 123 0xabcd1234> cell-property;
```

Square brackets are used to delimit arbitrary byte data, which is then entered in hex:

```javascript
binary-property = [01 23 45 67 89 ab cd ef]; binary-property = [01 23 45 67 89 ab cd ef];
```

A comma can be used to concatenate data with varying representations:

```javascript
0x12345678>; mixed-property = "a string", [01 23 45 67], 0x12345678;
```
Additionally, commas are used to create lists of strings:

```c
string-list = "red fish", "blue fish";
```

Aside about `/include/`

Similar to C's `#include` directive, the `/include/` directive results in simple textual inclusion, but a feature of the Device Tree compiler results in different usage patterns. Due to the fact that nodes are named, potentially with absolute paths, the same node may appear twice in a DTS file (and its inclusions). When this occurs, the nodes and properties are combined, with necessary interleaving and overwriting of properties (later values override earlier ones).

In the preceding example, the second appearance of `/node2` results in the addition of a new property to the original:

```
/node2 {
    an-empty-property;
    a-cell-property = <1 2 3 4>; /* each number (cell) is a uint32 */
    another-property-for-node2;
    child-node { 
        my-cousin = <&cousin>; 
    };
};
```
Thus, a single .dtsi file can overwrite or provide defaults for multiple locations in a tree.

**Labels and citations**

It is frequently necessary to refer to another part of the tree, and there are four ways to do so:

1. Strings defining paths

By analogy with a filesystem, paths should be self-explanatory - `/soc/i2s@7e203000` is the complete path to the I2S device in BCM2835 and BCM2836. Although it is trivial to construct a path to a property (for example, `/soc/i2s@7e203000/status`), the standard APIs do not do so; you must first locate a node and then select its properties.

**phantasms**

A phandle is a one-of-a-kind 32-bit integer that is assigned to a node's phandle property. You may also see a redundant, matching `linux,phandle` for historical reasons. Phantasms are sequentially numbered beginning with 1; 0 is not a valid phandle. They are typically allocated by the DT compiler whenever it encounters an integer context
reference to a node, typically in the form of a label (see below). References to nodes that use phandles are simply encoded as the corresponding integer (cell) values; no markup is used to indicate that they should be interpreted as phandles; this is application-defined.

3. Labeling

Just as a label in C gives a location in the code a name, a DT label gives a node in the hierarchy a name. When used in a string context (&node), the compiler converts references to labels to paths, and when used in an integer context (&node>, it converts them to phandles; the original labels do not appear in the compiled output. Take note that labels lack structure; they are simply tokens contained within a flat, global namespace.

4 Synonyms

Aliases are similar to labels, except that they appear as an index in the FDT output. They are contained within the /aliases node as properties, with each property mapping an alias name to a path string. While the aliases node is present in the source, the path strings are typically expressed as references to labels (&node) rather than being
written out completely. When DT APIs resolve a path string to a node, they typically start with the first character, treating paths that do not begin with a slash as aliases that must first be converted to a path using the /aliases table.

**Semantics of Device Trees**

How a Device Tree is constructed and how it is best used to capture the configuration of some hardware is a large and complex subject. While there are numerous resources available, some of which are listed below, several points warrant mention in this document:

The term "compatible properties" refers to the connection between the hardware description and the driver software. When an OS encounters a node with a compatible property, it searches its driver database for the best match. In most cases, this results in the driver module being automatically loaded in Linux, assuming it has been properly labeled and is not blacklisted.

The status property indicates whether or not a device is active. If the status is normal, normal, or not present, the device is enabled. Otherwise, status should be disabled, effectively disabling the device. It may be advantageous to
include devices in a.dtsi file with the status disabled. A derived configuration can then include that.dtsi file and set the required devices' status to okay.

**Overlays on Device Trees**

A modern SoC (System on a Chip) is a highly complex device; a complete Device Tree may run into the hundreds of lines. Taking that a step further and integrating the SoC with other components exacerbates the problem. To keep things manageable, particularly if related devices share components, it makes sense to place common elements in.dtsi files that can be included from potentially multiple .dts files.

When a system like the Raspberry Pi supports optional plug-in accessories such as HATs, the issue becomes more complicated. At the end of the day, each possible configuration requires a Device Tree to describe it, but once all the various With a variety of base models and a plethora of available accessories, the number of possible combinations begins to multiply rapidly. What is required is a mechanism for describing these optional components using a partial Device Tree and then building a complete
tree by starting with a base DT and adding a number of optional elements. This is possible, and these optional elements are referred to as "overlays." Unless you're interested in learning how to write overlays for Raspberry Pis, you're probably better off skipping ahead to Part 3: Using Device Trees on Raspberry Pi.

Fragments

A DT overlay is composed of multiple fragments, each of which is directed at a single node and its subnodes. Although the concept appears straightforward, the syntax appears strange at first:

```c
// Enable the i2s interface
/dts-v1/;
/plugin/;
/
{
    compatible = "brcm,bcm2835";

    fragment@0 {
        target = <&i2s>;
        __overlay__ {
            status = "okay";
            test_ref = <&test_label>;
            test_label: test_subnode {
                dummy;
            };
        };
    };
```
The compatible string indicates that this is for BCM2835, the base architecture for the Raspberry Pi SoCs; if the overlay makes use of Pi 4 features, brcm,bcm2711 should be used; otherwise, brcm,bcm2835 can be used for all Pi overlays. Then the first (and, in this case, the only) fragment appears. Fragments should be sequentially numbered beginning with zero. Failure to follow this procedure may result in the omission of some or all of your fragments.

Each fragment is composed of two components: a target property that specifies the node to which the overlay should be applied, and the __overlay__ itself, the body of which is appended to the target node. The preceding example can be interpreted as follows:

```c
/dts-v1/;
/plugin/;
/
{
    compatible = "brcm,bcm2835";
};
i2s {
    status = "okay";
    test_ref = &test_label;
    test_label: test_subnode {
        dummy;
    };
};
```
Indeed, with a sufficiently recent version of dtc, you can write it exactly as above and get identical output, but some homegrown tools do not yet support this format, so any overlays that you wish to include in the standard Raspberry Pi OS kernel should be written in the old format for the time being).

Merging that overlay with a standard Raspberry Pi base Device Tree (e.g. bcm2708-rpi-b-plus.dtb) would enable the I2S interface by setting its status to okay. However, if you attempt to compile this overlay using the following:

```
dtc -I dts -O dtb -o 2nd.dtbo 2nd-overlay.dts
```

You will encounter the following error:

```
i2s not found as a label or path
```

This is unsurprising, as there is no reference to the base.dtb or.dts file that would allow the compiler to locate the i2s label.

Attempting again, this time with the original example and the -@ option to allow unresolved references (along with -Hepapr to clean up the output):
dtc -@ -Heppapr -I dts -O dtb -o 1st-overlay.dts

If dtc returns an error regarding the third line, it lacks the required extensions for overlay work. Run `sudo apt install device-tree-compiler` again and the compilation should succeed this time. Notably, a suitable compiler is also included in the kernel tree as `scripts/dtc/dtc`, which is generated when the `dtbs make` target is used

**configure ARCH=arm dtbs**

Dumping the contents of the DTB file reveals the compiler's output:
Following the fragment, there are three additional nodes:

__symbols__ contains a list of the brands used in the overlay (in this case, test label), along with the path to the branded node. This node is critical in determining how unexplained symbols are handled.

__fixups__ contains a list of properties that map the names of unresolved symbols to lists of paths to cells within fragments that require patching with the phandle of the target node. Although the path is to the 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

The test ref property of __local fixups__ stores the locations of any references to labels that exist within the overlay. This is necessary because the merge program must ensure that the phandle numbers are sequential and unique.

In section 1.3, it is stated that "the original labels are not included in the compiled output," but this is not the case when the -@ switch is used. Rather than that, each label results in a property in the __symbols__ node,
which, like the aliases node, maps a label to a path. Indeed, the mechanism is so similar that when the Raspberry Pi loader attempts to resolve symbols, it searches the "aliases" node in the absence of a __symbols__ node. This was useful at one point because it allowed for the use of very old versions of dtc to build the base DTB files, but that is now history.

**Parameters for the Device Tree**

To avoid the need for numerous Device Tree overlays and to prevent peripheral users from editing DTS files, the Raspberry Pi loader now supports a new feature called Device Tree parameters. This enables minor modifications to the DT via named parameters, similar to how kernel modules receive parameters via modprobe and the kernel command line. The base DTBs and overlays, including HAT overlays, can expose parameters.

The DTS defines parameters by appending a __overrides__ node to the root. It contains properties whose names correspond to the selected parameter names and whose values are a sequence consisting of a
phandle (reference to a label) for the target node and a string indicating the target property; string, integer (cell), and boolean properties are all supported.

Parameters for strings

String parameters are declared in the following manner:

```
name = <&label>,"property";
```

Where appropriate values are substituted for the label and property. String parameters have the ability to expand, contract, or create their target properties.

Nota bene, the status properties are treated differently; values other than zero/true/yes/on are converted to the string "okay"", while zero/false/no/off becomes "disabled".

**Integer parameters**

Integer parameters are declared like this:

```
name = <&label>,"property.offset"; // 8-bit
name = <&label>,"property;offset"; // 16-bit
name = <&label>,"property:offset"; // 32-bit
name = <&label>,"property$offset"; // 64-bit
```
where label, property, and offset are replaced with appropriate values; the offset is clarified in bytes relative to the property's start (by default in decimal), and the preceding separator specifies the parameter's size. In contrast to previous implementations, integer parameters may refer to either non-existent properties or offsets beyond the end of an existing property.

**Boolean variables**

Device Tree stores boolean values as zero-length properties; if the property is present, it is true; if it is not present, it is false. They are defined as follows:

```c
boolean_property; // Set 'boolean_property' to true
```

Not that by not defining a property, it is assigned the value false. The following syntax is used to declare Boolean parameters:

```c
name = <&label>,"property?";
```

where appropriate values are substituted for the label and property.
Inverted booleans invert the input value before applying it in the same way as a regular boolean; they are declared similarly to regular booleans, except that they use the notation! to indicate the inversion:

\[name = \langle&\text{label}\rangle, "property!";\]

Boolean parameters can be used to create or delete properties, but they cannot be used to delete properties that already exist in the base DTB.

**Parameters for byte strings**

Byte string properties are sequences of bytes, for example, MAC addresses. They accept strings of hexadecimal bytes, separated by colons or not.

\[mac\_address = \langle&\text{ethernet0}\rangle, "local\_mac\_address"[;\]

The [ was chosen to match the DT syntax for declaring a byte string:

\[local\_mac\_address = [aa \text{ bb cc dd ee ff}];\]
parameters with multiple targets

Certain situations necessitate the ability to set the same value in multiple locations throughout the Device Tree. Rather than creating multiple parameters, it is possible to concatenate multiple targets into a single parameter, as follows:

```yaml
__overrides__ {
  gpiopin = &w1,"gpios:4",
  &w1_pins,"brcm,pins:0";
  ...
};
```

(from the w1-gpio overlay)

Note

It is even possible to use a single parameter to target properties of different types. An "enable" parameter could be connected to a status string, cells containing zero or one, and a proper boolean property.
Assignments in the literal sense

As demonstrated in 2.2.5, the DT parameter mechanism enables the patching of multiple targets from a single parameter, but its utility is limited by the requirement that the same value be written to all locations (except for format conversion and the negation available from inverted booleans).

By including embedded literal assignments, a parameter can write any value, regardless of the value supplied by the user.

Assignments are indicated by a := at the end of a declaration.

```
str_val = <&target>, "strprop=value"; // 1
int_val = <&target>, "intprop:0=42"; // 2
int_val2 = <&target>, "intprop:0=", <42>; // 3
bytes = <&target>, "bytestr[-b8:27:e8:01:23:45]"; // 4
```

The first two lines are self-explanatory, but line three is more interesting since the value appears as an integer (cell) value. While it is convenient for the DT compiler to evaluate integer expressions at compile time (especially
when macro values are used), the cell can also contain a reference to a label:

```c
// Force an LED to use a GPIO on the internal GPIO controller.
ex_led = <led1>, "gpio;0", "gpio;
ex_led1", "gpio;4");
```

Once the layer is applied, the label is resolved in the usual manner against the base DTB. Nota bene, it is a good practice to split multi-part parameters across multiple lines in this manner to make them easier to read - something that becomes even more necessary with the addition of cell value assignments.

Bear in mind that parameters have no effect unless they are used - a default value in a lookup table is ignored unless the parameter name is used without a value being assigned.

### Tables de recherche

Lookup tables enable the transformation of parameter input values prior to their use. They behave similarly to switch/case statements as associative arrays:

```c
phonetel = <inode>, "letter(a=alpha.b=bravo,c=charlie,d=e,=tango uniform!");
buse = <afreagment>, "target:0(0", "s1", "s101", ")");
```
It is the default value in the absence of a match, and
beginning or ending the list with a comma (or an empty
key=value pair anywhere) indicates that the unmatched
input value should be used unchanged; otherwise, an error
occurs.

Note

The comma separator within the table string following an
integer value in a cell is implicit; explicitly adding one
creates an empty pair (see above).

Note

Because lookup tables operate on input values and literal
assignments ignore them, the two cannot be combined -
characters following the closing in the lookup declaration
are treated as an error.

Parameters for overlay/fragmentation

The described DT parameter mechanism has a number of
drawbacks, including the inability to easily create arrays of
integers and the inability to create new nodes. One way to
circumvent some of these constraints is to include or exclude specific fragments on a conditional basis.

By renaming the __overlay__ node to __dormant__, a fragment can be excluded from the final merge process (disabled). The syntax for parameter declarations has been extended to allow the previously illegal zero target phandle to indicate that the following string contains fragment or overlay scope operations. Four operations have been carried out thus far:

```
+<0>    // Enable fragment <0>
+<0>    // Disable fragment <0>
+<0>    // Enable fragment <0> if the assigned parameter value is true, otherwise disable
+<0>    // Enable fragment <0> if the assigned parameter value is false, otherwise disable
```

Examples:

```
just_one  = <0>,"*1-2*"; // Enable 1, disable 2
conditional = <0>,"*3+4+"; // Enable 3, disable 4 if value is true, otherwise disable 3, enable 4.
```

This technique is used in the i2c-rtc overlay.

**Particular characteristics**

When a parameter is specified, a few real estate names receive special treatment. One you may have noticed already is status, which converts a boolean to either okay or disabled depending on whether it is true or false.
Assign different to the bootargs property adds to it rather than overwrites it - this is how kernel command line parameters can be added.

Reg specifies device addresses - the location of a memory-mapped hardware block, the address on an I2C bus, and so on. The names of child nodes should be qualified with their hexadecimal addresses, separated by @:

```plaintext
default bmp280@0x76 {
    reg = <0x77>;
    ...
};
```

When the reg property is assigned, the identify portion of the parent node's name is replaced by the assigned value. This can be used to avoid node name collisions when multiple instances of the same overlay are used - a technique that the i2c-gpio overlay employs.

The name estate is a pseudo-property; it should not appear in a DT, but when it is assigned, the name of its primary key is changed to the value assigned. As with the reg property, this can be used to assign unique names to nodes.
The file containing the overlay map

The introduction of the Pi 4, based on the BCM2711 SoC, brought numerous changes; some of these changes were interface additions, while others were interface modifications (or removals). There are some new overlays designed specifically for the Pi 4 that make no sense on older hardware, such as those that enable the new SPI, I2C, and UART interfaces, but others do not apply correctly despite controlling features that remain relevant on the new device.

As a result, a method for tailoring an overlay to multiple platforms with varying hardware is required. Supporting them all in a single.dtbo file would necessitate extensive use of hidden ("dormant") fragments and a switch to an on-demand symbol resolution mechanism to avoid failures caused by a missing symbol that is not required. A more straightforward solution would be to include a facility for mapping an overlay name to one of several implementation files, depending on the platform.

The overlay map, which is being rolled out in conjunction with the switch to Linux 5.4, is a file that the firmware
loads during boot up. It is written in DTS source format - overlay map. dts - and compiled to overlay map. dtb, which is then stored in the overlays directory.

This is a condensed version of the current map file (for the complete version, click here):

```
/ {
  vc4-kms-v3d {
    bcm2835;
    bcm2711 = "vc4-kms-v3d-pi4";
  };

  vc4-kms-v3d-pi4 {
    bcm2711;
  };

  uart5 {
    bcm2711;
  };

  pi3-disable-bt {
    renamed = "disable-bt";
  };

  lirc-rpi {
    deprecated = "use gpio-ir";
  };
};
```

Each node is associated with a base layer that requires special treatment. Each node's properties are either console names or one of a few special directives. Currently
supported platforms include bcm2835, which encompasses all Pis built around the BCM2835, BCM2836, and BCM2837 SoCs, and bcm2711, which encompasses the Pi 4B.

A platform name that contains no value (an empty property) indicates that the current overlay is compatible with it; for example, vc4-kms-v3d is compatible with the bcm2835 platform. A non-empty value for a platform specifies the name of an alternate overlay to load in place of the requested one; for example, requesting vc4-kms-v3d on BCM2711 loads vc4-kms-v3d-pi4. Any platform that is not included in the node of an overlay is incompatible with that overlay.

The second example node - vc4-kms-v3d-pi4 - could be deduced from the contents of vc4-kms-v3d, but that intelligence is applied during the file's construction, not during its interpretation.

If no platform is listed for an overlay, one of the following special directives may apply:

The renamed directive specifies the overlay's new name (which should be nearly identical to the original), but also logs a warning about the rename.
The deprecated directive includes a brief error message that will be logged when the common prefix overlay '...' is deprecated:

Bear in mind that only exceptions are required - the absence of an overlay node indicates that the default file should be used for all platforms. Debugging discusses obtaining diagnostic messages from the firmware. The utilities dtoverlay and dtmerge have been enhanced to include support for the map file:

dtmerge extracts the platform name from the base DTB's compatible string.

Although dtoverlay reads the compatible string from the live Device Tree located at /proc/device-tree, you can use the -p option to specify a different platform name (useful for dry runs on a different platform).

They both output errors, warnings, and debugging information to STDERR.
Examples

Here are some examples of various types of properties, along with the parameters that can be used to modify them:

```c
/ {
    fragment@0 {
        target-path = "/";
        __overlay__ {

            test: test_node {
                string = "hello";
                status = "disabled";
                bytes = /bits/ 8 <0x67 0x89>;
                ul6s = /bits/ 16 <0xabcd 0xef01>;
                u32s = /bits/ 32 <0xabcdef1234567890>
                u64s = /bits/ 64 <0x1234567890abcdef1234567890>
                bool; // Defaults to true
                bool2; // bool2 defaults to false
                mac = [01 23 45 67 89 ab];
                spi = <&spi0>;
            };
        };
    };

    fragment@1 {
        target-path = "/";
        __overlay__ {
            frag1;
        };
    };

    fragment@2 {
        target-path = "/";
        __dormant__ {
            frag2;
        };
    };
```
Additional examples can be found in the Raspberry Pi Linux GitHub repository, which hosts a sizable collection of overlay source files.

**Label exporting**

The firmware's overlay handling and the run-time overlay application that makes use of the dtoverlay utility consider labels defined in an overlay to be private to that overlay. This avoids the need to create globally unique names for labels (which keeps them concise), and it enables multiple instances of the same overlay to be used without clashing (provided some tricks are used - see Special properties).

However, there are times when it is extremely beneficial to be able to create a label with one overlay and use it with
another. Since 14 February 2020, firmware releases have included the ability to declare certain labels as global - the `__exports__` node:

```c
... public: ...

__exports__ {
    public; // Export the label 'public' to the base DT
}
};
```

When this overlay is applied, the loader removes all representations except those that have been shipped, in this case public, and rewrites the path to point to the fragment containing the label's target. After this one is loaded, subsequent overlays may refer to `&public`.

**Order of application of overlays**

Generally, the order in which fragments are applied is irrelevant, but for overlays that patch themselves (where the target of a fragment is a label within the overlay, referred to as an intra-overlay fragment), it becomes critical. In older firmware, fragments are applied in a strict top-to-bottom order. Since the 14th February 2020 release of firmware, fragments are applied in two passes:
The application and hiding of fragments that target other fragments begins with the application and hiding of fragments that target other fragments.

Following that, the standard fragments are applied.

This distinction is critical for runtime overlays, as step I is performed by the dtoverlay utility, while step (ii) is performed by the kernel (which is incapable of handling intra-overlay fragments).

**Device Trees on the Raspberry Pi DTBs, overlays, and the config.txt**

On a Raspberry Pi, the loader (one of the start.elf images) is responsible for combining overlays with an appropriate base device tree and then passing the kernel a fully resolved Device Tree. The base Device Trees are named bcm2711-rpi-4-b.dtb, bcm2710-rpi-3-b-plus.dtb, and so on, and are located alongside start.elf in the FAT partition (/boot from Linux). Nota bene, certain models (3A+, A, A+) will use their "b" equivalents (3B+, B, B+). This selection is made automatically and enables the use of the same SD card image in a variety of devices.
Take note that DT and ATAGs are mutually exclusive; passing a DT blob to a kernel that does not understand it will result in a boot failure. The firmware will always attempt to load the DT and pass it to the kernel, as all kernels since rpi-4.4.y require a DTB to function. This can be overridden by including device tree= in config.txt, which forces the use of ATAGs, which can be advantageous for simple "bare-metal" kernels.

[Previously, the firmware looked for a trailer appended to kernels by the mknlimg utility, but this feature has been deprecated.]

The loader now supports builds with bcm2835 defconfig, which specifies the BCM2835 support from upstream. This configuration will result in the creation of bcm2835-rpi-b.dtb and bcm2835-rpi-b-plus.dtb. If these files are copied along with the kernel, the loader will automatically attempt to load one of those DTBs. The loader supports the following config.txt directives for managing Device Tree and overlays:

When this overlay is applied, the loader removes all symbols except those that have been exported, in this case
public, and rewrites the path to point to the fragment containing the label's target. After this one is loaded, subsequent overlays may refer to &public.

**Order of application of overlays**

Generally, the order in which fragments are applied is irrelevant, but for overlays that patch themselves (where the target of a fragment is a label within the overlay, referred to as an intra-overlay fragment), it becomes critical. In older firmware, fragments are applied in a strict top-to-bottom order. Since the 14th February 2020 release of firmware, fragments are applied in two passes:

The application and hiding of fragments that target other fragments begins with the application and hiding of fragments that target other fragments.

Following that, the standard fragments are applied.

This distinction is critical for runtime overlays, as step I is performed by the dtoverlay utility, while step (ii) is performed by the kernel (which is incapable of handling intra-overlay fragments).
Device Trees on the Raspberry Pi DTBs, overlays, and the config.txt

On a Raspberry Pi, the loader (one of the start.elf images) is responsible for combining overlays with an appropriate base device tree and then passing the kernel a fully resolved Device Tree. The base Device Trees are named bcm2711-rpi-4-b.dtb, bcm2710-rpi-3-b-plus.dtb, and so on, and are located alongside start.elf in the FAT partition (/boot from Linux). Nota bene, certain models (3A+, A, A+) will use their "b" equivalents (3B+, B, B+). This selection is made automatically and enables the use of the same SD card image in a variety of devices.

Take note that DT and ATAGs are mutually exclusive; passing a DT blob to a kernel that does not understand it will result in a boot failure. The firmware will always attempt to load the DT and pass it to the kernel, as all kernels since rpi-4.4.y require a DTB to function. This can be overridden by including device tree= in config.txt, which forces the use of ATAGs, which can be advantageous for simple "bare-metal" kernels.
Previously, the firmware looked for a trailer appended to kernels by the mknlimg utility, but this feature has been deprecated.

The installer now supports builds with bcm2835 defconfig, which specifies the BCM2835 support from upstream. This configuration will result in the creation of bcm2835-rpi-b.dtb and bcm2835-rpi-b-plus.dtb. If these files are copied along with the kernel, the loader will automatically attempt to load one of those DTBs.

The loader supports the following config.txt directives for managing Device Tree and overlays:

```
  dtoverlay=acme-board
  dtparam=foo=bar,level=42
```

This instructs the installer to look for overlays/acme-board.dtbo in the firmware partition mounted on /boot by Raspberry Pi OS. It will then look for the parameters foo and level and assign them the specified values.

Additionally, the loader will look for an attached HAT with a programmed EEPROM and load the supporting overlay.
from there - either directly or by name from the "overlays" directory; this occurs automatically and without user intervention.

There are a number of ways to determine whether the kernel is utilizing Device Tree:

1. During bootup, the kernel message "Machine model:" contains a board-specific value, such as "Raspberry Pi 2 Model B," rather than "BCM2709."

2. The directory /proc/device-tree exists and contains subdirectories and files that precisely replicate the DT's nodes and properties.

The kernel will automatically search for and load modules that support the indicated enabled devices when using a Device Tree. As a result, by creating an appropriate DT overlay for a device, you prevent users from editing /etc/modules; all configuration is contained in config.txt, and in the case of a HAT, even that step is unnecessary. Notably, layered modules such as i2c-dev must still be loaded explicitly.
On the plus side, because platform devices are not created unless requested by the DTB, blacklisting modules that were previously loaded as a result of platform devices defined in the board support code should no longer be necessary. Indeed, the current Raspberry Pi OS images do not include any blacklist files (except for some WLAN devices where multiple drivers are available).

Parameters DT

As previously stated, DT parameters are a convenient way to make minor configuration changes to a device. Without the use of dedicated overlays, the current base DTBs support parameters for enabling and controlling the onboard audio, I2C, I2S, and SPI interfaces. When parameters are in use, they look like this:

```
dtparam=audio=on,i2c_arm=on,i2c_arm_baudrate=400000000,spi=on
```

Note

Multiple assignments may be placed on the same line, but the total length must not exceed 80 characters.
If you have an overlay that defines some parameters, they can be specified as follows:

```plaintext
dtoverlay=lirc-rpi
dtparam=gpio_out_pin=16
dtparam=gpio_in_pin=17
dtparam=gpio_in_pull=down
```

or concatenated to the overlay line in the following manner:

The parameters of an overlay are only valid until the next overlay is loaded. If both the overlay and the base export a parameter with the same name, the parameter in the overlay takes precedence; for clarity, it is recommended that you avoid doing so. To expose the parameter exported by the base DTB instead, use the following syntax to terminate the current overlay scope:

```plaintext
dtoverlay=
```

Labels and parameters unique to each board

Two I2C interfaces are available on Raspberry Pi boards. These are nominally divided into two sections: one for ARM and another for VideoCore (the "GPU"). On nearly all models, i2c1 is an ARM peripheral and i2c0 is a VC peripheral. Both are used to control the camera and read
the HAT EEPROM. However, two early revisions of the Model B reversed those roles.
The firmware creates some board-specific DT parameters to enable the use of a single set of overlays and parameters across all Pis. These include the following:

\texttt{i2c/i2c\_arm}

\texttt{i2c\_vc}

\texttt{i2c\_baudrate/i2c\_arm\_baudrate}

\texttt{i2c\_vc\_baudrate}

These really are aliases for \texttt{i2c0}, \texttt{i2c1}, \texttt{i2c0 baudrate}, and \texttt{i2c1 baudrate}. It is recommended that you only use \texttt{i2c vc} and \texttt{i2c vc baudrate} if you really need to - for example, if you are programming a HAT EEPROM (which is better done using a software I2C bus using the \texttt{i2c-gpio overlay}) (which is better done using a software I2C bus using the \texttt{i2c-gpio overlay}). Enabling \texttt{i2c vc} can stop the Pi Camera or 7" DSI display functioning correctly.

For people writing overlays, the same aliasing has been applied to the labels on the I2C DT nodes. Thus, you should write:
Overlays that currently use the mathematical variants will be updated to use the new aliases.

**HATs and Tree of Devices**

A Raspberry Pi HAT is an add-on board with an embedded EEPROM that is compatible with Raspberry Pi models equipped with a 40-pin header. The EEPROM contains any DT overlays required to enable the board (or the name of an overlay to load from the filing system), which may also expose parameters. The firmware loads the HAT overlay automatically after the base DTB, which means that its parameters are accessible until any other overlays are loaded or the overlay scope is ended using dtoverlay=. If you wish to disable the HAT overlay loading for any reason, place dtoverlay= before any other dtoverlay or dtparam directive.

Tree of Dynamic Devices
The RPi kernels support dynamic loading of overlays and parameters as of Linux 4.4. Compatible kernels manage an overlay stack that is layered on top of the base DTB. Changes are immediately reflected in /proc/device-tree and may result in the loading of modules and the creation and destruction of platform devices.

The term "stack" is critical here; overlays can be added and removed only at the top of the stack; changing anything further down the stack requires that everything on top of it be removed first.

There are several new commands for managing overlays, including the following:

**overlay is a command.**

Overlay is a command-line utility that loads and unloads overlays while the system is running, as well as listing and displaying information about available overlays. To obtain usage information, run overlay -h:

```
Usage:
doverlay <overlay> [param=val]...  
doverlay -D [idx]         Add an overlay (with parameters) 
doverlay -S [overlay]     Remove an overlay (by name, index or the last) 
doverlay -R [overlay]     Remove from an overlay (by name, index or all) 
doverlay -L               List active overlays/params 
doverlay -A               List all overlays (marking the active) 
doverlay -h               Show this usage message 
doverlay -h <overlay>     Display help on an overlay 
doverlay -h <overlay> [param]... Or its parameters 
where <overlay> is the name of an overlay or 'dtparams' for dtparams 

Options applicable to most variants: 
-d <dir> Specify an alternate location for the overlays [defaults to /boot/overlays or /flash/overlays] 
-v               Verbose operation
```
Unlike the config.txt equivalent, all overlay parameters must be included in the same command line - the dtparam command is only for base DTB parameters.

Two points to keep in mind:
1. Command variants that modify the kernel's state (adding and removing objects) require root privilege, so the command may need to be prefixed with sudo.
2. Only run-time overlays and parameters can be unloaded - an overlay or parameter applied by the firmware is "baked in," meaning it is not listed by dtoverlay and thus cannot be removed.

The command dtparam

dtparam creates and loads an overlay that has a similar effect to the config.txt dtparam directive. It is largely equivalent to dtoverlay with an overlay name of - in terms of usage, but there are a few distinctions:
1. dtparam will display the help information for all known DTB parameters. Using dtparam -h, you can access help for the dtparam command.
2. When specifying a parameter to be removed, only index numbers are permitted (not names).
3. Not all Linux subsystems respond to runtime device additions – I2C, SPI, and sound devices all work, but others do not.

**Writing guidelines for runtime-capable overlays**

Although this area is sparsely documented, the following tips have accumulated:

- A device object is created or deleted when a node is added or removed, or when the status of a node changes from disabled to enabled or vice versa. Be cautious - the lack of a "status" property indicates that the node is enabled.
- Avoid creating a node within a fragment that overwrites an existing node in the base DTB - the kernel will rename the new node to distinguish it from the existing node. Create a fragment that targets an existing node if you wish to modify its properties.
- ALSA does not prevent the unloading of its codecs and other components while they are in use. Removing an overlay may result in a kernel exception if it deletes a codec that a sound card is still using. Through experimentation, it was discovered that devices are deleted from the Because the order of the fragments in the overlay is reversed, placing the card node after the component nodes enables an orderly shutdown.
Caveats
The ability to load overlays at runtime is a relatively recent addition to the kernel, and there is currently no accepted method for doing so from userspace. By concealing the details of this mechanism behind commands, the goal is to protect users from changes that might occur if a different kernel interface becomes standard.

Certain overlays perform better than others at run-time. Components of the Device Tree are only used during boot; altering them via an overlay has no effect.

Applying or removing certain overlays may result in unexpected behavior; therefore, it should be done cautiously. This is one of the reasons why sudo is required.

Unloading the overlay for an ALSA card can cause the system to stall if something is actively using ALSA - as demonstrated by the LXPanel volume slider plugin. To enable the removal of sound card overlays, the lxpanelctl utility has been extended with two new options - alsastop and alsastart - which are invoked before and after overlays are loaded or unloaded, respectively, by the auxiliary scripts dtoverlay-pre and dtoverlay-post.
While removing an overlay does not result in the unloading of a loaded module, it may result in the reference count of some modules dropping to zero. Using `rmmod -a` twice will result in the unloading of unused modules.

Overlays must be removed in the reverse order in which they were applied. While the commands will allow you to remove an earlier one, they will also remove and re-apply all intermediate ones, which may have unintended consequences.

Only the top-level Device Tree nodes and the children of a bus node will be probed. Additionally, for nodes added during operation, the bus must register for notifications of child additions and removals. There are, however, exceptions to this rule that create confusion: The kernel scans the entire tree explicitly for certain device types - clocks and interrupt controllers are the two primary ones - in order to initialize them early (for clocks) and/or in a specific order (for interrupt controllers).

This search mechanism occurs only during boot and thus does not work for nodes added during run-time by an overlay. It is therefore recommended that overlays be placed at the root of the tree with fixed-clock nodes unless
it is guaranteed that the overlay will not be used at runtime.

overlays and parameters that are supported

As it would be impractical to document each overlay individually here, refer back to the README file included with the overlay.dtbo files in /boot/overlays. It is constantly updated with new information and changes.

**Troubleshooting \ Debugging**

The loader will bypass missing overlays and incorrect parameters, but if serious errors occur, such as a missing or corrupt base DTB or a failed overlay merge, the loader will fall back to a non-DT boot. If this occurs, or if your settings do not behave as expected, it is worthwhile to check for loader warnings or errors:

```
sudo vcdbg log msg
dtdebug=1 can be added to config.txt to enable additional debugging.
```

You can create the following human-readable representation of DT's current state:

```
dtc -l fs /proc/device-tree
```
This is useful for visualizing the effect of overlays being merged onto the underpinning tree.

If kernel modules fail to load properly, verify that they are not blacklisted in /etc/modprobe.d/raspi-blacklist.conf; when using Device Tree, blacklisting should be unnecessary. If that reveals nothing suspicious, you can also verify that the module exports the appropriate aliases by performing a search.

/lib/modules/<version>/modules.alias for the compatible value. Otherwise, your driver is probably missing either:

of_match_table = xxx_of_match,

OR

MODULE_DEVICE_TABLE(of, xxx_of_match);

In the absence of that, either depmod struggled or the updated modules were not equipped on the target filesystem.

Overlays are validated using dtmerge, dtdiff, and ovmerge.

Along with the dtoverlay and dtparam commands, there is a utility for overlaying a DTB called dtmerge. To begin using it, you must first acquire your base DTB, which can be acquired in one of two ways:
a. generate it from the current state of the device tree in /proc/device-tree:

\texttt{dtc -I fs -O dtb -o base.dtb /proc/device-tree}

These include any layers and criteria you have applied thus far, either in config.txt or via runtime loading, which may or may not be desired. Alternatively

b) copy it from the /boot source DTBs. This excludes overlays and parameters, but also any other firmware modifications. To facilitate testing of all overlays, the dtmerge utility will create some board-specific aliases ("i2c arm", for example), but this means that the merged DTB will contain more differences than expected. The solution to this is to create a copy using dtmerge:

\texttt{dtmerge /boot/bcm2710-rpi-3-b.dtb base.dtb – (the - indicates an absent overlay name).}

You can now try applying an overlay or parameter:

\texttt{dtmerge base.dtb merged.dtb - sd overclock=62}

\texttt{dtdiff base.dtb merged.dtb}

which will return:
You can also compare different overlays or parameters.

dtmerge base.dtb merged1.dtb /boot/overlays/spi1-1cs.dtb

dtmerge base.dtb merged2.dtb /boot/overlays/spi1-2cs.dtb

dtdiff merged1.dtb merged2.dtb
Another DT utility is included in the Utile repo: ovmerge. In contrast to dtmerge, ovmerge merges files and applies overlays in their original form. Due to the fact that the overlay is never compiled, labels are preserved, resulting in a more readable result. Additionally, it includes a number of additional features, such as the ability to list the order of file inclusion.

**Constraints on the Device Tree**

If you have very specific requirements that the default DTBs do not address, or if you simply want to experiment with writing your own DTs, you can instruct the loader to load an alternate DTB file as follows:

```
device_tree=my-pi.dtb
```

**Disabling the use of the Device Tree**

Device Tree support is required in Pi Linux kernels since the switch to the 4.4 kernel and the inclusion of more upstream drivers. However, for bare metal and other operating systems, the procedure for disabling DT usage is as follows:
device_tree=

to config.txt.

**Shortcuts and syntax variants**
The loader understands a few shortcuts:

\[\text{dtparam=i2c\_arm=on}\]
\[\text{dtparam=i2s=on}\]

can be abbreviated as:

\[\text{dtparam=i2c,i2s}\]

[i2c is an alias for i2c arm, and the =on assumption is made]. Additionally, it accepts the extended forms device tree overlay and device tree param.

Additional DT commands are included in config.txt.

Device tree address This is used to override the firmware's address for loading the smartphone tree (not dt-blob). By default, the firmware will select an appropriate location.

device tree end This limits the packed device tree (exclusively). By default, the device tree can grow indefinitely, which is almost certainly what is required.
dtdebug If this value is greater than zero, enable some additional logging for the firmware's device tree processing.

Enable uart Allow access to the primary/console UART (ttyS0 on a Pi 3, ttyAMA0 otherwise - unless swapped with an overlay such as miniuart-bt). If ttyAMA0 is the primary UART, enable uart defaults to 1 (enabled), otherwise to 0 (disabled).

This is necessary to prevent the core frequency from changing, rendering ttyS0 unusable, and thus enable uart=1 implies core freq=250 (unless force turbo=1). Because this can have a negative impact on performance in some cases, it is disabled by default.

Overlay prefix Specifies the subdirectory/prefix from which overlays should be loaded - by default, this is "overlays/". Keep an eye out for the trailing "/". If desired, you can add something after the final "/" to give each file a prefix, though this is unlikely to be necessary.

Additional terminals can be monitored by the DT; see section 3 for details.
Additional assistance

If you've read this document and are still unable to resolve a Device Tree issue, assistance is available. Generally, the author can be found on Raspberry Pi forums, most notably on the Device Tree forum.

Kernel Command-Line Interface

During boot, the Linux kernel accepts a command line of parameters. On the Raspberry Pi, this command line is defined in a file named cmdline.txt in the boot partition. This is a straightforward text file that can be edited with any text editor, for example, Nano.

`sudo nano /boot/cmdline.txt`

Note

We have to use sudo to edit anything in the boot partition, and all parameters in cmdline.txt must be on the same line (no carriage returns).

Using cat /proc/cmdline, you can view the command line passed to the kernel at boot time. It will not be identical to the one in cmdline.txt because the firmware may modify it prior to launching the kernel.
Options for the Command Line

There are numerous kernel command-line interface parameters, several of which are kernel-defined. Others are defined by kernel code, for example, the Plymouth splash screen system.

Typical Entries

console: This variable specifies the serial console. Typically, there are two entries:

console=serial0,115200

console=ttty1

root: specifies the root filesystem's location, for example, root=/dev/mmcblk0p2 indicates video production card block 0 partition 2.

rootfstype: specifies the filesystem type that the rootfs will use, for example, rootfstype=ext4.

Elevator: specifies the scheduler for I/O operations to use. Elevator=deadline indicates that the kernel places a time constraint on all I/O operations in order to avoid request starvation.
Quiet: sets the default kernel log level to KERN WARNING, which suppresses all log messages except those containing critical information during boot.

Entries in the FKMS and KMS modes

Through an entry such as: The bootloader automatically creates a preferred resolution and overscan setting.

`video=HDMI-A:1:1920x1080M@60,margin_left=0,margin_right=0,margin_top=0,margin_bottom`

This standard entry can be modified by manually duplicating it in `/boot/cmdline.txt` and modifying the margin parameters as necessary. Additionally, as demonstrated in the standard Linux framebuffer documentation, it is necessary to add rotation and reflect parameters. By default, the margin_* options are set based on the presence of overscan entries in `config.txt`. By adding `disable fw kms setup=1` to `config.txt`, the firmware can be prevented from making any KMS-specific changes to the command line.

The following is an example of an entry:
Additional Entries (not exhaustive)

Splash: instructs the boot to use the Plymouth module's splash screen.

Plymouth. Ignore-serial-consoles: By default, enabling the Plymouth module prevents boot texts from appearing on any serial consoles that may be present. This flag instructs Plymouth to dismiss all serial consoles, restoring visibility to boot messages as they would be in the absence of Plymouth.

dwc_otg.lpm_enable=0: turns off Link Power Management (LPM) in the dwc_otg driver; the dwc_otg driver is the driver for the USB controller built into the processor used on Raspberry Pi computers.

Note

This controller is disabled by default on the Raspberry Pi 4 and is only attached to the type C power input jack; the Pi
4's type A USB ports are influenced by a separate USB controller that is unaffected by this setting.

dwc otg.speed: controls the speed of the built-in USB controller on Raspberry Pi computers. dwc otg.speed=1 configures it to operate at full speed (USB 1.0), which is slower than high speed (USB 2.0). This option should not be enabled except when troubleshooting USB device problems.

Note

This controller is disabled by default on the Raspberry Pi 4 and is connected only to the type C power connector; the type A USB ports on the Pi 4 are driven by a separate USB controller that is unaffected by this setting.

**Smc95xx**.turbo mode: enables/disables the turbo mode for the wired networking driver. **smc95xx.turbo** mode=N disables the turbo mode.

U bhid.mousepoll: specifies the mouse polling interval. If you have problems with a slow or erratic wireless mouse, setting this to 0 might help: usbhid.mousepoll=0.
UART Configuration

On the Raspberry Pi, two types of UART are available: PL011 and mini UART. The PL011 is a capable UART that is broadly compatible with the 16550 standard, whereas the mini UART has a more limited feature set.

All UARTs on the Raspberry Pi operate at 3.3V only; connecting them to 5V systems will cause damage. Connecting to 5V systems requires the use of an adaptor. Alternatively, various third-party vendors offer low-cost USB to 3.3V serial adaptors.

Raspberry Pi Zero, One, Two, and Three

Each Raspberry Pi Zero, 1, 2, and 3 has two UARTs:

4 and 400 MHz Raspberry Pi

The Raspberry Pi 4B and 400 models include four additional PL011s that are disabled by default:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART0</td>
<td>PL011</td>
</tr>
<tr>
<td>UART1</td>
<td>mini UART</td>
</tr>
<tr>
<td>UART2</td>
<td>PL011</td>
</tr>
<tr>
<td>UART3</td>
<td>PL011</td>
</tr>
<tr>
<td>UART4</td>
<td>PL011</td>
</tr>
<tr>
<td>UART5</td>
<td>PL011</td>
</tr>
</tbody>
</table>
CM, CM 3, CM 3+ and CM 4
Each Compute Module from the first generation, as well as Compute Module 3 and Compute Module 3+, has two UARTs, whereas Computer Module 4 has six UARTs as described above.

UARTs are disabled by default on all compute module models and must be explicitly enabled using a device tree overlay. Additionally, you must specify which GPIO pins to use, for instance:

\texttt{dtoverlay=uart1,txd1\_pin=32,rxd1\_pin=33}

**UART primary**

On the Raspberry Pi, a single UART is chosen to be available on GPIO 14 (transmit) and 15 (receive) - this is the main UART. This is also the UART on which a Linux device may be present by default. Nota bene, GPIO 14 corresponds to pin 8 on the GPIO header, whereas GPIO 15 corresponds to pin 10.

**UART secondary**

Normally, the secondary UART is not connected to the GPIO connector. By default, on models that include this controller, the supplementary UART is linked to the
Bluetooth side of the combined wireless LAN/Bluetooth controller.

UARTs, primary and secondary

**UART primary**

On the Raspberry Pi, a single UART is chosen to be present on GPIO 14 (transmit) and 15 (receive) - this is the primary UART. This is also the UART on which a Linux console may be present by default. Nota bene, GPIO 14 corresponds to pin 8 on the GPIO header, whereas GPIO 15 corresponds to pin 10.

**UART secondary**

Normally, the secondary UART is not connected to the GPIO connector. By default, on models that include this controller, the secondary UART is connected to the Bluetooth side of the combined wireless LAN/Bluetooth controller.

UARTs, primary and secondary

The following table summarizes how the first two UARTs were assigned:

<table>
<thead>
<tr>
<th>Mini UART PL011 (UART0) model first</th>
<th>Zero Raspberry Pi</th>
</tr>
</thead>
</table>

**Note**
By default, the mini UART is disabled, regardless of whether it is designated primary or secondary.

Linux-based devices running on the Raspberry Pi OS:

<table>
<thead>
<tr>
<th>Linux device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/ttyS0</td>
<td>mini UART</td>
</tr>
<tr>
<td>/dev/ttyAMA0</td>
<td>first PL011 (UART0)</td>
</tr>
<tr>
<td>/dev/serial0</td>
<td>primary UART</td>
</tr>
<tr>
<td>/dev/serial1</td>
<td>secondary UART</td>
</tr>
</tbody>
</table>

/dev/serial0 and /dev/serial1 are symbolic links which point to either /dev/ttyS0 or /dev/ttyAMA0.

**Frequency of the Mini-UART and the CPU Core**

To use the mini UART, the Raspberry Pi must be configured to run at a repaired VPU core clock frequency. This is because the mini UART clock is coupled to the VPU core clock, which means that as the core clock frequency changes, the UART baud rate changes as well.

The enable uart and core freq options in config.txt can be used to modify the behavior of the mini UART. The following table summarizes the combinations that are possible:
Mini UART connected to the system's core clock

Result

<table>
<thead>
<tr>
<th>Mini UART set to</th>
<th>core clock</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary UART</td>
<td>variable</td>
<td>mini UART enabled</td>
</tr>
<tr>
<td>primary UART</td>
<td>fixed by setting <code>enable_uart=1</code></td>
<td>mini UART enabled, core clock fixed to 250MHz, or if <code>core_freq=250</code> is set, the VPU turbo frequency.</td>
</tr>
<tr>
<td>secondary UART</td>
<td>variable</td>
<td>mini UART disabled</td>
</tr>
<tr>
<td>secondary UART</td>
<td>fixed by setting <code>core_freq=250</code></td>
<td>mini UART enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primary UART</th>
<th>Default state of <code>enable_uart</code> flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>mini UART</td>
<td>0</td>
</tr>
<tr>
<td>first PL011 (UART0)</td>
<td>1</td>
</tr>
</tbody>
</table>

**Disabling the Serial Console in Linux**

By default, the Linux console is assigned the primary UART. To repurpose the primary UART, you must rearrange Raspberry Pi OS. This can be accomplished through the use of raspi-config:

1. Run `raspi-config` by typing `sudo raspi-config`.
2. Click on the third option - Interface Options.
3. Select the P6 - Interface Configuration option.
4. In response to the prompt Would you like to be able to access a login shell via serial? 'No' as an answer
5. In response to the prompt Would you like to enable the serial port hardware? 'Yes' in response
6. Exit raspi-config and reboot the Pi to apply the changes.

**Enabling Linux's Early Console**
Although the Linux kernel initializes the UARTs relatively early in the boot process, this is still after some critical pieces of infrastructure have been established. Without access to the kernel log messages from that time period, a failure in those early stages can be difficult to diagnose.

To enable earlycon support for one of the UARTs, modify cmdline.txt with one of the following options, depending on which UART is primary:

For the Raspberry Pi 4, 400, and the Compute Module 4:

```
earlycon=uart8250,mmio32,0xfe215040
earlycon=p1011,mmio32,0xfe201000
```

For Pi 2, Pi 3 and Compute Module 3:

```
earlycon=uart8250,mmio32,0x3f215040
earlycon=p1011,mmio32,0x3f201000
```

For Pi 1, Pi Zero and Compute Module:

```
earlycon=uart8250,mmio32,0x20215040
earlycon=p1011,mmio32,0x20201000
```

The baudrate is set to 115200bps by default.

**Note**
Selecting the incorrect early console can result in the Pi failing to boot.

**Device Tree and UARTs**

Numerous UART Hardware Tree overlay definitions are available on the kernel's GitHub repository. Disable-bt and minuart-bt are the 2 different most useful overlays. deactivate disconnects the Bluetooth technology and assigns the primary UART to the first PL011 (UART0).

Additionally, you must disable the system service that initializes the modem, using `sudo systemctl disable hciuart`, to prevent it from connecting to the UART.

minuart-bt configures the Bluetooth function to use the mini UART and assigns the primary UART to the first PL011 (UART0). Take note that this may result in a reduction in the maximum baud rate that can be used (see mini UART limitations below). Additionally, either `force turbo=1` or `core freq=250` must be used to set the VPU core clock to a fixed frequency.

The overlays uart2, uart3, uart4, and uart5 enable the additional four UARTs on the Raspberry Pi 4. Additional
UART-specific overlays are included in the folder. For more information on Device Tree overlays, see /boot/overlays/README or run dtoverlay -h layer for explanations and usage information.

To apply a Device Tree overlay, you add a line to the config.txt file. Take note that the -overlay.dts suffix has been omitted from the filename. For instance
dtovelay=disable-bt

PL011 and a miniature UART

There are some distinctions between PL011 and mini-UART UARTs.
The mini-FIFOs UART's are smaller. When combined with poor of flow control, this increases the likelihood of characters being lost at higher baudrates. Additionally, it is less capable than a PL011, owing to the baud rate's relationship to the VPU clock speed.
The mini UART's specific shortcomings in comparison to a PL011 are as follows:

• There is no detection of breaks
• Detection of no framing errors
• There is no parity bit.
• There is no timeout for receiving.
• There are no DCD, DSR, DTR, or RI signals present.
Additional information about the mini UART is available in the SoC peripherals document.

Icons for Firmware Warnings

The Raspberry Pi firmware will occasionally display a warning icon on the display to indicate an issue. At the moment, three icons are available for display.

Warning: Under voltage

If the Raspberry Pi's power supply voltage falls below 4.63V (5%), the following icon is displayed.

Warning: Excessive Heat (80-85°C)

If the SoC temperature is between 80 and 85 degrees Celsius, the following icon is showcased. The ARM core(s) will be throttled down in an attempt to cool the core.
Warning: Excessive Temperature (greater than 85°C)

When the SoC temperature exceeds 85°C, the following icon is displayed. The ARM core(s) and GPU will be throttled back in an attempt to cool the core.

LED Flashing Warning Codes
If a Raspberry Pi fails to boot or must be shut down for any reason, an LED will frequently flash a set number of times to indicate what happened. To indicate the exact status, the LED will blink for a number of long flashes (0 or more), followed by a series of short flashes. Typically, the pattern will repeat itself after a two-second pause.

<table>
<thead>
<tr>
<th>Long flashes</th>
<th>Short flashes</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Generic failure to boot</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>start elf not found</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>Kernel image not found</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>SDRAM failure</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>Insufficient SDRAM</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>In HALT state</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Partition not FAT</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Failed to read from partition</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Extended partition not FAT</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>File signature/hash mismatch - Pi 4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>SPI EEPROM error - Pi 4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>SPI EEPROM is write protected - Pi 4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>SD card error - Pi 4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Unsupported board type</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Fatal firmware error</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>Power failure type A</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>Power failure type B</td>
</tr>
</tbody>
</table>

**Keeping your Raspberry Pi secure**

Security is critical for your Raspberry Pi. Security flaws expose your Raspberry Pi to hackers, who can then use it without your consent.

The security level you require is determined by the intended use of your Raspberry Pi. For instance, if you are
simply connecting your Raspberry Pi to your home network via a router equipped with a firewall, it seems to be quite secure by default.

However, if you wish to connect your Raspberry Pi directly to the internet, either via a direct connection (unlikely) or by allowing certain protocols (e.g. SSH) to pass through your router's firewall, you must make some basic security changes.

Even if your network is protected by a firewall, it is prudent to take security seriously. This document will discuss several methods for enhancing the security of your Raspberry Pi. Please keep in mind that this list is not exhaustive.

**Default Password Change**

For each Raspberry Pi operating Raspberry Pi OS, the default username and password are used. Therefore, if you gain access to a Raspberry Pi and ensure that these settings have not been altered, you will have root access to that Raspberry Pi.

Therefore, the first step is to change the password. This can be accomplished using either the raspi-config application or the command line.

```
sudo raspi-config
```
Choose option 2 and follow the on-screen instructions to reset the password.

However, raspi-config does nothing more than start the command prompt `passwd` application, which can be started directly from the command line. As an alternative, you can type in and confirm your new password.

`passwd`

**Modification of your Username**

Naturally, you can increase the security of your Raspberry Pi even further by changing your username as well. Because all Raspberry Pis ship with the default username `pi`, changing it immediately increases the security of your Raspberry Pi.

To establish a new user, enter the following:

`sudo adduser alice`

You will be compelled to create a new user's password.

The new user's home directory will be located at `/home/alice/`.

To add them to the sudo group, which grants them sudo permissions in addition to all other required permissions:

`adm,dialout,cdrom,sudo,audio,video,plugdev,games,users,input,netdev,gpio,i2c,spi alice sudo usermod -a -G`
You can verify that your permissions are correct (i.e., by using sudo) by performing the following:
su - alice
If it succeeds, you can be certain that the new account is added to the sudo group.
After verifying that the new account is operational, you can delete the pi user. To accomplish this, you must first change the autologin user to your new user alice as follows:

```
sudo raspi-config
```
Choose alternative 1, S5 Boot / Auto login, and confirm rebooting. Please keep in mind that certain aspects of the existing Raspberry Pi OS distribution require the presence of the pi user. If you're not sure whether you'll be impacted, leave the pi user in place. Efforts are being made to reduce reliance on the pi user.
Type the following to delete the pi user:

```
sudo deluser pi
```
This command deletes the pi user but leaves the /home/pi directory intact. If necessary, you can also use the orders below to delete the pi user's home folder simultaneously. Due to the fact that the contents of this folder will be permanently deleted, ensure that any necessary data is stored elsewhere.
**sudo deluser -remove-home pi**

This command will generate a warning indicating that the gathering pi is no longer active. However, because the deluser command deletes both the pi user and group, the caution can be safely ignored.

**Create sudo Require the Use of a Password**

By prefixing a command with sudo, it is run as a superuser, which does not require a password by default. This is not a general problem. However, if your Pi is exposed to the internet and is exploited in some way (for example, via a webpage exploit), the attacker will be able to change anything that requires superuser credentials, unless you have sudo set to require a password.

To require a passcodes for sudo, enter:

**sudo visudo /etc/sudoers.d/010_pi-nopasswd**

and modify the pi entry (or any other usernames with superuser rights) as follows:

```
pi ALL=(ALL) PASSWD: ALL
```

The file will then be checked for syntax errors. If there are no errors, the file is saved and you are returned to the shell prompt. If errors are detected, you will be prompted with the question 'what now?' On your keyboard, press the 'enter' key to bring up a list of options. You're probably
going to want to use 'e' for '(e)dit sudoers file again', so that you can edit the file and resolve the issue.

Note
Choosing option 'Q' will save the file with any remaining syntax errors, rendering the sudo command inaccessible to any user.

**Updating the Raspberry Pi operating system**
A current distribution includes all available security updates, so you should update your version of Raspberry Pi OS to the most recent version.
If you connect to your Raspberry Pi via SSH, it may be worthwhile to add a cron job that updates the ssh-server specifically. The following command, perhaps as a daily cron job, will ensure that you are always up to date with the latest SSH security fixes, regardless of your normal update process.
install openssh-server via apt-get

SSH Security Enhancement
SSH is a popular method of remotely accessing a Raspberry Pi. SSH authentication requires a username/password pair by default, but there are ways to make this more secure. Utilizing key-based authentication is an even more secure method.
Enhancing the security of usernames and passwords

The most critical step is to create a strong password. If your Raspberry Pi is connected to the internet, your password must be extremely strong. This will assist in avoiding dictionary attacks or similar attacks. Additionally, you can allow or deny specific users via the sshd configuration.

/etc/ssh/sshd config sudo nano
Add, edit, or append the following line to the end of the file to include the usernames you wish to allow to log in:
AllowUsers bob alice
Additionally, you can use DenyUsers to prevent specific usernames from logging in:
jane john – DenyUsers
Following the change, you must restart the sshd service with sudo systemctl restart ssh or reboot to ensure that the changes take effect.

Utilizing a key-based authentication scheme.
Two cryptographically secure keys are referred to as key pairs. One is for private use, while the other is for public use. They can be used to verify a client's connection to an SSH server (in this case the Raspberry Pi).
The client generates two cryptographically linked keys. While the private key should never be shared, the public key is permissible. When a link is requested, the SSH server copies the public key and uses it to send the client a challenge message, which the client will encrypt using the private key. If the server is able to decrypt this message back to the original challenge message using the public key, the client's identity can be verified.

In Linux, the ssh-keygen command is used to generate a key pair; the keys are stored by default in the .ssh folder in the user's home directory. The private key is named id_rsa, and the public key is named id_rsa.pub. The key will be 2048 bits in length; decrypting such a key would take an extremely long time, making it extremely secure. If the situation necessitates it, you can create longer keys. Take note that you should only run the generation process once; if you do so again, any previously generated keys will be overwritten. Anything that relies on those old keys must be updated to use the new ones.

During key generation, you will be prompted for a passphrase: this adds an additional layer of security. Leave this blank for the time being.
The public key must now be transferred to the server: see
Take a copy of your public key and save it to your
Raspberry Pi.
Finally, we must disable password logins, relying entirely
on key pairs for authentication.

`sudo nano /etc/ssh/sshd_config`

Three lines must be altered to no, when they're not already:

```
ChallengeResponseAuthentication no
PasswordAuthentication no
UsePAM no
```

Should save file and either reboot or sudo service ssh
reload the ssh system.

**Configure a Firewall**

There are numerous Linux firewall solutions available. The
majority of them rely on the fundamental iptables project
to perform packet filtering. This project is a wrapper
around the Linux network filtering system. iptables is
installed but not configured by default on Raspberry Pi OS.
Configuring it can be challenging, and one project that
offers a more straightforward interface than compatibles is
ufw, which stands for 'Relatively simple Fire Wall'. This is Ubuntu's default firewall tool, and it's very easy to install on your Raspberry Pi:

```
sudo apt install ufw
sudo ufw enable
```

To disable the firewall, and disable start up on boot, use:

```
sudo ufw disable
```

Allow a particular port to have access (we have used port 22 in our example):

```
sudo ufw allow 22
```

Denying access on a port is also very simple (again, we have used port 22 as an example):

```
sudo ufw deny 22
```

You can also specify which service you are allowing or denying on a port. In this example, we are denying tcp on port 22:

```
sudo ufw deny 22/tcp
```
You can specify the service even if you do not know which port it uses. This example allows the ssh service access through the firewall:

```
sudo ufw allow ssh
```

The status command lists all current settings for the firewall:

```
sudo ufw status
```

The rules can be quite complicated, allowing for the blocking of specific IP addresses, specifying the direction in which data transmission is allowed, or limiting the number of connection attempts, for example, to aid in the defeat of a Denial of Service (DoS) attack. Additionally, you can specify which devices will be affected by the device rules (e.g. eth0, wlan0). For complete details, please refer to the ufw man page (man ufw), but here are some examples of more sophisticated commands.

Limit login attempts via tcp on the ssh port: This prevents connection if an IP address attempts to connect six times or more in the last 30 seconds:

```
sudo ufw limit ssh/tcp
```

Deny access to port 30 from IP address 192.168.2.1
**sudo ufw deny from 192.168.2.1 port 30**

**Failing to install fail2ban**

If you're using your Raspberry Pi as a server, such as an ssh or web server, your firewall will have intentional 'holes' to allow server traffic to pass through. Fail2ban may be beneficial in these instances. Fail2ban is a Python-based scanner that explores the Raspberry Pi's log files for suspicious activity. It detects instances such as multiple brute-force login attempts and can notify any installed firewall to prevent further login attempts from suspect IP addresses. It eliminates the need to manually check log files for malicious activity and then update the firewall to prevent them (via iptables).

Use the following command to install fail2ban:

**sudo apt install fail2ban**

Fail2ban generates a folder /etc/fail2ban during installation, which contains a configuration file named jail.conf. This must be copied to jail.local in order for it to function. This configuration file contains a set of default options, as well as options for monitoring services for abnormal behavior. Examine/modify the ssh rules as follows:
In the jail.local file, add the following section. This section may already exist in some versions of fail2ban; please update it if it does.

```plaintext
[ssh]
enabled  = true
port     = ssh
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 6
```

As you can see, this section is named ssh, is enabled, examines the ssh port, filters for malicious activity using the sshd variables, decodes the /var/log/auth.log for fraudulent attacks, and allows six retries before the detection threshold is reached. By inspecting the default segment, we can see that the default action for banning is as follows:

```plaintext
# Default banning action (e.g. iptables, iptables-new, # iptables-multiport, shorewall, etc) It is used to define # action_* variables. Can be overridden globally or per # section within jail.local file
banaction = iptables-multiport
```

When the detection threshold is reached, the Fail2ban system will execute the /etc/fail2ban/action.d/iptables-
multiport.conf file. There are numerous action configuration files available. Multiport imposes a total ban on all access to all ports.

If you want to permanently ban an IP address after three failed attempts, you can set the maxretry value in the [ssh] section to a negative value and the ban time to a negative value:

```
[ssh]
enabled = true
port   = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = -1
```

**Configuring Blanking of the Screen**

You can customize your Raspberry Pi to run a screen saver or to turn off the display entirely.

**At the Console**

When not connected to a diagrammatical desktop, Raspberry Pi OS will automatically blank the screen after ten minutes of inactivity, such as mouse movement or key presses.

The current setting, in seconds, can be viewed by using the following:
To modify the consoleblank configuration, edit the kernel command line as follows:

**sudo nano /boot/cmdline.txt**

A single line of text is contained in the file /boot/cmdline.txt. Add console blank=n to make the console inactive after n seconds. For example, console blank=300 will cause the controller to become inactive after 300 seconds, or 5 minutes. Ensure that you include your console blank option in the cmdline.txt file's single line of text. Set console blank=0 to disable screen blanking.

Additionally, you can disable screen blanking using the raspi-config tool. Not that the screen blanking setting in raspi-config also applies when the graphical desktop is running.

**On the Workstation**

After ten minutes without user input, Raspberry Pi OS will blank the graphical desktop. This can be disabled by modifying the 'Screen Blanking' option in the Raspberry Pi Configuration tool's Preferences menu. Notably, the
'Screen Blanking' option also controls the state of the screen when the graphical desktop is not running. Additionally, there is a graphical screensaver that can be installed as follows:

```
sudo apt install xscreensaver
```

This process may take a few moments. Once installed, the Screensaver application is located in the Preferences menu; it provides numerous options for configuring the screensaver, including completely disabling it.

**Disabling HDMI**

If you wish to completely disable the video display, you can use the vcgencmd command.

```
vcgencmd display_power 0
```

*Video will not come back on until you reboot or switch it back on:*

```
vcgencmd display_power 1
```

**The boot directory**

In a basic Raspberry Pi OS installation, the boot documents are saved on the SD card's first partition, which is
formatted as FAT. This implies that it can be read on devices running Windows, macOS, or Linux.

When the Raspberry Pi is energized on, it loads various files from the boot partition/folder to initialize the various processors, and then the Linux kernel is booted. After booting Linux, the boot partition is mounted as /boot.

**Contents of the Boot Folder**

**bootcode.bin**

This is the bootloader, which is packed by the SoC during boot and performs some very basic configuration before loading one of the start*.elf files. On the Raspberry Pi 4, bootcode.bin is not used because it has been replaced by boot code stored in the onboard EEPROM.

**start.elf, start x.elf, start db.elf, start cd.elf, start4.elf, start4x.elf, start4db.elf, start4x.elf, start4cd.elf, start4db.elf, start4x.elf, start4cd.elf, start4db.elf**

These are binary blobs (firmware) that are loaded onto the SoC's VideoCore and then take control of the boot process. The start.Elf file contains the basic firmware, start x.elf contains the camera drivers and codec, start db.elf contains the debug version of the firmware, and start cd.elf contains a stripped-down version of the firmware that lacks support
hardware blocks such as codecs and 3D, and is intended for 
use when gpu mem=16 is specified in config.txt. The 
config.txt file contains additional information on how to 
use these.

start4.elf, start4x.elf, start4cd.elf, and start4db.elf are 
specific firmware files for the Raspberry Pi 4.
These are linker files that correspond to the previous 
section's start*.elf files.

cmdline.txt
The kernel command line that is passed to the kernel 
during bootup.
Contains numerous configuration parameters for 
configuring the Raspberry Pi. Consult the config.txt file.

issue.txt
Some text-based housekeeping information about the 
distribution, including the date and git commit ID.

ssh or ssh.txt ssh.txt
SSH will be empowered on boot if this file is present. The 
contents are irrelevant; it can be vacant. Otherwise, SSH is 
disabled by default.

wpa_supplicant.conf
It is possible if the equipment is capable of it). Modify the
nation code and network section to suit your needs. The
wireless/headless section additional information on how to
use this file.

**Files containing Device Trees**
There are numerous Device Tree blob files that end in.dtb.
These files contain the hardware definitions for the various
Raspberry Pi models and are used during boot to configure
the kernel according to the detected Pi model.

**Kernel Configuration Files**
The boot folder will contain a variety of kernel image files
that will be used to boot the various Raspberry Pi models:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Processor</th>
<th>Raspberry Pi model</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel.img</td>
<td>BCM2835</td>
<td>Pi 0, Pi 1</td>
<td></td>
</tr>
<tr>
<td>kernel7.img</td>
<td>BCM2836, BCM2837</td>
<td>Pi 2, Pi 3</td>
<td>Later Pi 2 uses the BCM2837</td>
</tr>
<tr>
<td>kernel7l.img</td>
<td>BCM2711</td>
<td>Pi 4</td>
<td>Large Physical Address Extension (LPAE)</td>
</tr>
<tr>
<td>kernel8.img</td>
<td>BCM2837, BCM2711</td>
<td>Pi 2, Pi 3, Pi 4</td>
<td>Beta 64-bit kernel, Raspberry Pi 2 with BCM2836 does not support 64-bit.</td>
</tr>
</tbody>
</table>

For 32-bit processes (i.e. everything except kernel8.img),
lscpu reports the architecture as armv7l, and for 64-bit
systems as aarch64. The l in the armv7l case indicates that
the architecture is little-endian, not LPAE, as the l in the
kernel7l.img filename indicates.
The Folder for Overlays

Device Tree overlays are stored in the overlays sub-folder. These are used to configure various hardware devices connected to the system, such as the Raspberry Pi Touch Display and third-party sound boards. These overlays are specified in config.txt — for more information, see 'Device Trees, Overlays, and Parameters, Part 2'.
CHAPTER EIGHT

Troubleshooting

OCCASIONALLY, THINGS DO NOT GO PERFECTLY. The more complicated the gadget, the more complicated the difficulties that may arise—and the Pi is an extraordinarily sophisticated device.

Fortunately, some of the most frequent issues are simple to diagnose and resolve. This chapter examines one of most prevalent causes of the Pi's misbehavior and how to resolve them.

Diagnostics for the keyboard and mouse

Perhaps the most frequently encountered issue with the Raspberry Pi is when specific characters are repeated on the keyboard. For instance, if the command start displays on the screen as s, it will naturally fail to execute when the Enter key is pushed. Typically, there are two possible explanations why a USB keyboard does not work properly when attached to the Raspberry Pi: It is either taking too
much power or its own chipset is interfering with the Pi's USB hardware.

Check the paperwork for your computer or the marking on the underside to determine whether it has a milliamp power rating (mA). When the keyboard is in use, this is the amount of power the laptop attempts to draw out from USB port.

The USB ports on the Raspberry Pi are not as powerful as those on a full-size laptop or desktop computer. This can be a problem with keyboards that include integrated LED illumination, which consumes far more power than a regular keyboard.

If you suspect that your USB keyboard is consuming excessive power, connect it to a powered USB hub rather than straight to the Pi. This allows the keyboard to be powered by the hub's power supply unit rather than the Pi. Alternatively, replace the keyboard with a more energy-efficient model.

The repeating-letter issue can possibly be attributed to an insufficient power source for the Pi, as discussed in the following section, "Power Diagnostics."
Unfortunately, compatibility is a more difficult issue to diagnose. While the vast majority of keyboards function flawlessly with the Pi, a handful exhibit unusual symptoms. These include the following:

sporadic reaction to the repeating-letters condition, or even complete shutdown of the Pi. Occasionally, these issues do not manifest themselves until additional USB computers are connected to the Pi. If your keyboard functioned normally until another USB device, specifically a USB bluetooth adapter, was connected, you may have an interoperability issue or your USB dongle may be taking too much power.

If possible, switch the keyboard with a different model. If the new laptop works, it is possible that your old one is incorrect with the Pi. Visit the eLinux wiki at http://elinux.org/Rpi USB Keyboards for a list of supported keyboards, however keep in mind that this list is user-generated and far from exhaustive.

The same advice about verifying compatibility in advance applies to mouse problems. While the majority of USB mice and trackballs operate flawlessly, a few are incompatible with the Raspberry Pi's USB hardware. This
typically manifests as a jerky or unresponsive mouse cursor, but it can occasionally result in the Raspberry Pi failing to load or crashing at random intervals. If you're in the market for a new mouse, the Linux wiki site http://elinux.org/RPi USB Mouse devices contains a list of models known to operate with the Pi.

**Electrical Diagnostics**

Numerous issues with the Raspberry Pi can be attributed to an insufficient power source. While the Model A consumes a maximum of 500mA, the high-performance Raspberry Pi 3 consumes up to 1,200mA (1.2A), a value that increases when additional accessories are added.

Not all USB power adapters, despite their labeling, are meant to deliver this much power.

If you're experiencing sporadic troubles with your Pi—particularly if it works until you add something to a USB connection or initiate a processor-intensive function such as video playback—the likelihood is that the power supply you're using is insufficient. The Raspberry Pi's power LED functions as an integrated voltage tester, informing you if the power supply you're using falls short of the standard
required for stable operation. If the power LED is flashing or is not illuminated, your power source is less than 4.65V—significantly less than the 5V USB standard—and should be changed.

Another indicator of insufficient power is the presence of a rainbow-colored square in the upper right corner of the display. If your power consumption is close to the limit, you may notice that this square appears and disappears when you vary the amount of power drawn by the system, for example, by running something that utilizes the graphics processor or by adding electronics to the GPIO header.

If you want to get a better understanding of the amount of power being received by your Pi, the simplest option is to get a USB power meter. A simple sort of multimeter, a USB power meter is used to test the voltage and amperage of your USB electrical supply and the Raspberry Pi.

A USB cable compatible with your USB power meter's input should be linked to the USB power meter's input, and its output should be connected to the Raspberry Pi through a micro-USB cable. When the USB power meter is linked to power, it begins reading statistics on the quality
of the power coming from the power supply. These statistics include the voltage, which is commonly denoted on the display by a V for volts, and the current, which is denoted by an A for amps (see Figure 4-1).

![USB power meter](image)

**Figure 4-1:** A USB power meter, reading the power flowing into a Raspberry Pi 2

the USB power meter's voltage reading should be between 4.65V and 5.2V.

If it is less than 4.65V, the Pi is not receiving enough power.

Change the USB adapter and verify that it can produce 700mA or more for lower-powered Raspberry Pis such as the Model A and Pi Zero, 1.8A or more for the A+, B+, and Raspberry Pi 2, and 2.5A or more for the Raspberry Pi 3.
Be cautious of low-cost power supply and ultra-thin micro-USB cables: They can have incorrect labeling and fail to deliver the advertised current. Genuine branded power supply, such as the official Raspberry Pi type, are rarely affected by this issue, but unbranded devices, which are frequently offered as suitable converters, should be avoided.

The current reading can be used to detect whether you are approaching the maximum output of your power source. If the readout indicates "0.62A," for example, this indicates that you are now using 620mA (milliamps, thousandths of an amp), which is somewhat more than the rated output of a 500mA power source but safely less than the output of a 1A or more power supply. While a Raspberry Pi should draw less than 500mA in typical operation, this amount can grow when additional hardware such as Wi-Fi dongles, onboard screens, or wireless keyboards and mouse is connected. A USB power meter is the simplest and most secure way to monitor the amount of current being consumed.
Diagnostics on Display

Although the Pi is designed to connect with virtually any HDMI, DVI, or composite video display device, it may not function properly when plugged in. For instance, you may discover that your image has been pushed to the side or is not fully displayed, or that it is only visible as a postage-stamp-sized cutout in the center of the screen or in black and white—or that it is completely absent.

To begin, determine the type of device to which the Raspberry Pi is connected. This is especially critical if you're connecting the Pi to a TV through the composite connection. Different nations have their own standards for television video, which means that a Raspberry Pi configured for one region may not function in another. This is the most common reason for a Raspberry Pi that displays black-and-white video. Chapter 7, "Advanced Raspberry Pi Configuration," describes how to alter this parameter.

When using the HDMI output, the display type is often detected automatically. If you're connecting the Pi to a computer monitor through an HDMI to DVI or VGA converter, something can occasionally go wrong. Typical
symptoms include static in the form of snow, missing areas of the image, or no display at all. To resolve the issue, take note of the resolution and refresh rate of the connected monitor and then skip to Chapter 7 to learn how to manually adjust them.

Another issue is a picture that is either too huge or too little, either missing bits at the screen's edge or sitting in the center of a large blank border. This issue is caused by a setting called overscan, which is enabled when the Pi is connected to a television in order to avoid showing on sections of the display that are obscured behind a bezel/surround. As with other display-related settings, Chapter 7 teaches you how to reduce — or even totally disable — overscan.

**Diagnostics During Boot**

The most frequently encountered reason for a Raspberry Pi to fail to boot is an issue with the SD (or microSD) card.

Unlike a desktop or laptop computer, the Raspberry Pi runs entirely on files stored on the SD card. If the Pi is unable to communicate with the card, it will not display
anything else on the screen or demonstrate any signs of life.

Currently, work is being done to guarantee that a wider variety of cards work well with the Pi. Before entirely abandoning it, examine if a new version of your preferred Linux operating system is available. (For additional information about distributions, see Chapter 1, "Meet the Raspberry Pi." )

If you have been overclocking your Raspberry Pi (see Chapter 6, "The Raspberry Pi Configuration Tool"), it may not boot properly. Hold down the Shift key while the boot messages display on-screen to temporarily disable the overclocked and run the Pi at its default speed.

**Diagnostics of Networks**

`ifconfig` is the most valuable tool for detecting network problems. If you're connecting over a wireless network, go on to Chapter 5, "Network Configuration," for information on a comparable utility for those devices. Otherwise, continue reading.

`Ifconfig` is a powerful utility for controlling and configuring the Raspberry Pi's network ports. It was designed to offer
information about connected network ports. To use the tool in its simplest form, simply type its name in the terminal:

    ifconfig

When used in this manner, ifconfig returns information about all available network ports. (see Figure 4-2). There are two ports on the regular Raspberry Pi Model B, Model B+, and Raspberry Pi 2: an ethernet Lan port on the right side of the board and a virtual loopback interface that enables the Pi's programs to communicate with one another. Meanwhile, the Raspberry Pi 3 includes a third port for the built-in wireless network adapter.
Ifconfig's output is broken down into two parts:

Link encapsulation—The network's encapsulation type, which on the Model B will either read Ethernet for the actual network port (often designated eth0) or Regional Loopback for the digital loopback adaptor (usually named lo).

Hwaddr—The hexadecimal representation of the network interface's Media Access Control (MAC) address. This is unique for each device on the network, and each Raspberry Pi is assigned its own MAC address at the factory.
inet addr—The network interface's Internet Protocol (IP) address. This is how you locate the Raspberry Pi on the network if it is running a network-accessible service, such as a web server or file server.

Bcast—The network's broadcast address to which the Pi is connected. Any traffic directed to this address will be received by all network devices.

The network mask specifies the maximum size of the network to which the Raspberry Pi is connected. For the majority of residential users, this will display as 255.255.255.0.

MTU—Maximum transmission unit size, or the maximum size of a single packet of data before it must be broken into numerous packets.

RX—This part gives information about the network traffic that was received, including the amount of faults and dropped packets. If you begin to observe errors in this section, there is a problem with the network.

TX—This component contains the same details as the RX section, but for packets that have been transferred. Again, any problems logged here indicate a network issue.
Collisions—When two systems on the network attempt to communicate at the same time, a collision occurs, requiring both systems to retransmit their packets. While a small number of collisions is acceptable, a large number signals a network issue.

txqueuelen—The transmission queue's length, which is typically set to 1000 and rarely needs to be changed.

RX bytes, TX bytes—A summary of the quantity of traffic passed over the network interface.

If you're having network issues with the Pi, consider disabling and then re-enabling the network interface. The simplest way to accomplish this is by the use of two tools named ifup and ifdown.

If the network is operational but not functioning properly—for example, if the inet addr section of ifconfig is empty—start by disconnecting the network port. Enter the following command in the terminal:

```bash
ifdown eth0 sudo
```

After disabling the network, ensure that the cable is securely attached at both ends and that the network device
to which the Pi is connected (hub, switch, or router) is powered on and operational. Then, using the following command, restore the interface:

**ifup eth0 sudo**

The `sudo ping` command can be used to test the networking by sending data to a remote machine and waiting for a response. If everything is functioning properly, you should see a response similar to that illustrated in Figure 4-3. If not, you may need to establish your network manually, as described in Chapter 5, "Network Configuration."
Figure 4-3: The result of a successful test of the network, using the `sudo ping` command.
CHAPTER NINE

An Introduction to Python
The first half of the raspberry Pi's name comes from a long tradition of naming new computing systems after fruits (from classic minicomputers like the Acorn, Apricot, and Tangerine to more recognizably modern brands like Apple and BlackBerry), but the second half comes from the Python programming language.

Python is a programming language. Python was invented in the late 1980s at the National Research Institute for Mathematics and Computer Science by Guido van Rossum as a replacement for the ABC programming language. Python has increased in popularity since its inception, owing to what is considered as a clear and expressive syntax that was built with the goal of making code readable.

Python is a programming language with a high level of abstraction. This means Python programs written in mainly recognizable English, sending commands to the Raspberry Pi in a way that is simple to understand and follow. Low-level languages, such as assembly, are closer to
how a computer "thinks," but are nearly difficult for a
human to understand without prior training. Python's
high-level nature and simple syntax make it a fantastic tool
for anyone learning to program. It's also the language that
the Raspberry Pi Foundation recommends for individuals
who want to move on from Scratch (explained in Chapter
10, "An Introduction to Scratch") to more "hands-on"
programming.

Python is an open source programming language that is
freely available for Linux, OS X, and Windows computers.
This cross-platform support implies that software built for
the Raspberry Pi can be run on computers running
practically any other operating system, with the exception
of programs that need Pi-specific hardware such as the
GPIO interface. To learn more about how Python can be
used to address this port, click here.

Example 1: Hello World

The simplest approach to learn a whole new computer
language is to develop a project that produces "Hello
World!" on the screen, as you discovered in Chapter 10. In
Scratch, you just dragged and dropped prewritten code blocks, but in Python, you must write the complete program by hand.

At its core, a Python project is simply a text file that contains clear instructions for the computer to execute. Any text editor can be used to generate this file. For example, nano can be used if you prefer working at a console or in a terminal window, while Leafpad can be used if you prefer a graphical user interface (GUI). Another option is to use an integrated development environment (IDE) like IDLE, which has Python-specific features that aren't available in a conventional text editor, such as syntax checking, debugging, and the ability to launch your program without leaving the editor. This chapter shows you how to generate Python files with IDLE, but the IDE or editor you use for programming is entirely up to you. This chapter also includes directions for running your newly produced files from the terminal, which can be used with any text editor or IDE.

Open Python 2 (IDLE) from the Programming option on the Raspbian distribution's desktop environment to start the Hello World project. If you don't have IDLE, start a
new document with your preferred text editor and skip the rest of this section. IDLE starts up in Python shell mode by default (see Figure 11-1), so anything you input in the first window will be performed right away. To create a new Python program that can be run later, go to the File menu and select New File from the drop-down menu. All Python programs should begin with a line known as a shebang, which gets its name from the # and! characters at the start of the line. This line instructs the operating system to look for the Python interpreter in a certain location. This is not required for programmes that are executed directly by calling the program's filename, but it is essential for applications that are run from within IDLE or that explicitly call Python at the terminal.

The first line of your program should be as follows to ensure that it works regardless of where the Python executable is installed:
This line instructs the operating system to look for Python in the $PATH environment variable, which records the place(s) of files that can be launched as programs in Linux. Python should function on any Linux distribution installed on the Pi. The $PATH variable holds a list of directories that contain executable files and is used to locate applications when you type their names into the console or terminal window.

You should use Python's print statement to print a message. This statement, as the name implies, prints text to an output device, which is usually the console or terminals window from which the program is run. It is easy
to use. Text written within quotation marks after the word print will be printed to the output device device.

In your new project, type the following line.

If you're using IDLE instead of a plain text editor to write the example program, you'll notice that the text is multicolored (see Figure 11-2, where colours are represented as differing shades of grey in the print edition). This is known as syntax highlighting, and it's a feature found in IDEs and more powerful text editors. To make the program easier to grasp at a glance, syntax highlighting alters the color of parts of the text as according their function. It also makes it simple to discover so-called syntax problems, such as failing to include an end-quote in a print instruction or failing to comment out a remark. Syntax highlighting isn't required in this example;
nevertheless, in larger projects, it can be a valuable aid for locating problems.

Use the File menu to save your program as helloworld.py before running it. If you're using IDLE, the extension is applied automatically. If you're using a text editor, make sure to save the file with the extension.py rather than.txt. Although Python is intelligent enough to run the program even if it's stored with a different file extension, this extension signifies that the file includes Python code.

Depending on whether you're using IDLE or a text editor, you'll need to run the file in a different way. Simply select Run Module from the Run menu or press the F5 key on your keyboard in IDLE. This returns IDLE to the Python
shell window, where the application is executed. The
message Hello, World! should then show in blue on the
screen (see Figure 11-3). If not, double-check your syntax,
making sure you have quote marks at the start and end of
the statement on the print line.

You'll need to launch a command prompt from the
Accessories menu on the desktop if you wrote the
helloworld.py script in a text editor. If you saved the file
somewhere other than your home directory, you'll need to
change directories with the cd command (see Chapter 3,
"Linux System Administration"). Once you're in the correct
directory, type the following to start your program:
helloworld.py in Python This instructs the system software to launch Python and then execute the helloworld.py file. Python, unlike the Python shell in IDLE, terminates when the file is finished and returns you to the terminal. However, the end consequence is the same: the message Hello, World! is printed to standard output (see Figure 11-4).

Getting Python Programs to Run To launch a Python program, you must first tell the Python program to download the file. However, because of the shebang line at the head of the file, it is possible to run the file without first calling Python. This is a convenient technique to create your own terminal tools: after copying the Python program to a location in the system's $PATH environment variable, the Python program may be called simply by typing its name.
To begin, you must notify Linux that the Python file should be tagged as executable—a property that indicates that the file is a program. This property isn't automatically set to secure the state from malware obtained from the Internet, so only executable files will run. Use the chmod command (covered in detail in Chapter 3) to make the helloworld.py file executable by typing: helloworld.py chmod +x

Now type the following into the command prompt to launch the application directly:

Despite the fact that you didn't type python helloworld.py, the helloworld.py program should run exactly as if you had written python helloworld.py. Only the complete path to the program—/home/pi/helloworld.py—or the current directory (using./ as the path) can be used to launch it. To make the file available in the same way as any other desktop command, use the following command to copy it to /usr/local/bin: /usr/local/bin/sudo cp helloworld.py

Nonprivileged users cannot write to the /usr/local/bin directory for security reasons, therefore the sudo prefix is essential. The helloworld.py file, which is located in /usr/local/bin and is included in the $PATH variable, can be executed by simply typing its name from any directory.
Try going to a separate directory and then entering `helloworld.py` to run the application. You can rename your custom-made programs to remove the .py file extension to make them appear more like native utilities. Simply type:

```
sudo mv /usr/local/bin/helloworld.py /usr/local/bin/helloworld.py /usr/local/bin/helloworld.py /usr/local/bin/helloworld.py /usr/local/bin
```

**Example 2: Comments, Inputs, Variables, and Loops**

The Hello World program is a practical and moderate intro to a language, but it isn't particularly fascinating. By its very nature, it merely covers the fundamentals and neglects to teach some of the principles necessary for developing effective and engaging programs. The next example, on the other hand, makes use of some of the fundamental Python features for creating interactive programs.

Begin by starting a new blank document in IDLE or your text editor, like in Example 1, and then running the program with the shebang line:
This line isn't necessarily required unless program is going to be made operational, but it does no harm and is a great method to establish, as previously said.

Then, if you need to download the file at a later time, add a comment to the application to provide context. Note that this, like other code lines that terminate in a symbol, must be inserted as a single line: # Example 2: A program written in Python from the User's Guide for the Raspberry Pi

Excluding the shebang line (which can only appear on the first line of the program), anything after a hash symbol is handled as a comment in Python. When Python encounters a comment, it ignores it and moves on to the next line. It's a good idea to comment your code since, while you may understand what a certain bit of code does now, things may not be as clear six months down the road. Comments also make code more accessible, and if you choose to share your code with others, they will be able to comprehend what each section is supposed to do thanks to your comments. It's not actually necessary to include remarks in simple programs; but, like adding the shebang line, it's a nice habit to develop. Comments can be on their own line, as in the previous comment, or at the end of a
line, in which case Python will continue running the code line until it hits the hash symbol.

Then, using the following line, ask the user for his or her name: raw input = userName

("Can you tell me your name?")

This short line does a great deal. The first part, userName =, instructs Python to establish a new variable called userName, which serves as a storage space for data. The equals symbol instructs Python to set the variable to whatever follows. In this situation, though, what follows isn't just a piece of data, but a new function: raw input.

This tool is intended to receive string (text) input from the keyboards and publish a message to the default output so that the user knows what to enter. This keeps the software basic since it requires a preceding line with a print instruction to print a prompt advising the user what to type. If you don't leave a space at the end of the prompt, the user's input will start right after the question mark.

The software can now get clever with the user's name safely kept in the userName variable. Use the following line to greet the user: userName, print "Welcome to the software,"
This line displays the ability to issue the contents of variables, which is a supplementary feature of the print instruction introduced in Example 1. The first half of this print command publishes everything between the two quote marks, but the second comma instructs print that more should be printed to the same line. Python understands that inputting the variable name userName tells it to print the contents of that variable, resulting in a message personalized to the user's name.

This sample application is a straightforward but user-friendly calculator. Unlike Example 1, it keeps running until the user orders it to stop. This is accomplished by the use of a loop, much like in Scratch. Begin the loop by typing the two lines below:

while goAgain == 1: goAgain == 1: goAgain == 1: goAgain == 1: goAgain =

The first line declares a new variable, goAgain, and initializes it to 1. The second line starts the loop by telling Python that it should loop through the following code as long as the goAgain variable is equal to 1. The next several lines must be indented, which is accomplished by adding four space at the beginning of each line. These spaces tell
Python which lines are inside and which are outside of the loop. The spaces are automatically inserted in IDLE; if you're using a text editor, remember to manually insert the spaces.

**Why ==?**

Previously, you set the value of a variable with a single equals symbol. The while loop, on the other hand, employs two. When two equals symbols are placed next to each other, an evaluation is performed, which compares the value of one variable to the value of the next variable. Instead, a single equals symbol sets the variables to the value after it.

Other evaluations exist in addition to the double-equals, which is correct only if the variable exactly matches the given value: > indicates greater than, < indicates less than, >= indicates greater or equal to, = indicates or less than equal to, and != indicates not equal to.

You can use these evaluation symbols to regulate the flow of a program using Boolean logic principles. Refer to Chapter 10 for further information on Boolean logic.
A calculator takes two integers as input and does an arithmetic operations on them in its most basic form. To get your calculator to work, get the numbers from the user using the lines below:

```python
firstNumber = int(input("Type the first number: "));
secondNumber = int(input("Type the second number: ")); 
```

These lines not only ask for two numbers using the raw input instruction, and they also use int. The int command, which stands for integer, tells Python to handle input as a numbers rather than a string. Obviously, since a calculator application will not be calculating words, this is critical.

The software can do its calculations using the two numbers as variables. Add, subtract, and multiply the two values with the following lines, then transmit the outcome to the user:

```python
secondNumber, "equals", print firstNumber, "added to", 
```
firstNumber + secondNumber print "minus" firstNumber, "equals" secondNumber

firstNumber – secondNumber print firstNumber, "multiplied by," secondNumber, "equals," firstNumber, "multiplied by," secondNumber, "equals,"

secondNumber * firstNumber

While the plus and minus symbols are used in the addition and subtraction operations, the * symbol is used in multiplication. There are no styling spaces between both the quotation marks, as well. This is because when Python writes integers and strings together, it immediately adds spaces where they are needed. Finally, note that no division operation (represented by the / symbol) is performed. This is due to the fact that the sample calculator application only accepts integers, which are only allowed to be whole numbers with no decimal places or fractions.

Although the calculating portion of the programme is now complete, it will continue to run indefinitely because Python has no way of knowing when it is time to escape the loop. Add the following line to give a means for the user to terminate the program: goAgain = int(raw
input("Type 1 to enter more numbers, or any other number to quit: ") goAgain = int(raw input("Type 1 to enter more numbers, or any other number to quit: ")) goAgain = int(raw input

The goAgain variable, which controls the while loop, can now be changed by the user. The loop will run again if the user enters the number 1, because the goAgain variable is still set to 1. If the user enters any other number, however, the evaluation is no longer true (goAgain is no longer equal to 1), and the loop ends.

The completed program should look like this (remember to enter everything marked with on a single line):

#!/usr/bin/env python

Example 2: From the Raspberry Pi User Guide, a Python application raw input("What is your name?") userName = raw input("What is your name?") "Welcome to the program," print "Welcome to the program," goAgain = 1 while goAgain == 1:
firstNumber = int(raw input("Type the first number: "))
firstNumber = int(raw input("Type the first number: "))
firstNumber = int

secondNumber = int(raw input("Type the second number: "))
secondNumber = int(raw input("Type the second number: "))
secondNumber = int secondNumber, "equals",
print firstNumber, "added to",
firstNumber + secondNumber print "minus" firstNumber, "equals" secondNumber

firstNumber – secondNumber print firstNumber, "multiplied by," secondNumber, "equals," firstNumber, "multiplied by," secondNumber, "equals,"
goAgain = int(raw input("Type 1 to enter more numbers, or any other number to quit: ")) firstNumber * secondNumber

Choose Launch Module from the Run menu in IDLE or type python calculator.py at the terminal to run the program. When prompted, enter your username; then enter the numbers you want to calculate (see Figure 11-5) until you're bored, then type any number other than 1 to
Visit the official Python Simple Programs wiki page at https://wiki.python.org/moin/SimplePrograms for more small programs that introduce fundamental Python concepts.

**Example 3: Gaming with pygame**

To demonstrate Python's capabilities, this example builds a fully playable arcade game based on the classic game of Snake or Nibbles.

The pygame Python package is used to accomplish this.
pygame is a collection of Python modules designed by Pete Shinners to extend the language's capabilities—specifically, to make it easier to write games in Python.

Music, graphics, and even network functions are all included in each pygame module, which are all necessary in a modern game.

Although you may make a game in Python without using the pygame package, it is much easier if you use the existing code.

If you're using Raspbian, the pygame library already is installed and ready to use.

For many distributions, the pygame source files can be acquired from the official pygame website at www.pygame.org/download.shtml.

Installation instructions can be found on the same page.

A pygame project is created similarly to any other Python project.

In IDLE or a text editor, add the following shebang line to the top of a new blank page: python #!/usr/bin/env
The following step is to tell Python that this program uses the pygame modules.

You do this by using an import statement, which tells Python to load an external module (a different Python file) and make it available to the current program.

Write the following two lines into your new project to import the necessary modules:

```
pygame, sys, time, and random are imported from pygame.
```

```
pygame, sys, time, and random are all imported from pygame.
```

The first line imports the pygame module, along with the Python modules sys, time, and random, which will all be utilized in this program.

The second line in the associated code directs Python to load all of the instructions from the pygame.locals module as native instructions. Normally, a component is called by typing its name, a full stop, and the name of the orders from within the module, but the second line in the corresponding code directs Python to load all of the
instructions from the pygame.locals module as native instructions.

As a result, typing will be reduced while using these instructions.

Other module names, such as pygame.clock (which is not to be confused with pygame.locals), will necessitate full typing.

Type the following two lines to get pygame ready to use in the example application:

```python
pygame.init()
pygame.time.
fpsClock = Clock()
```

The first line tells pygame to start up, while the second sets up a new variable named fpsClock to manage the game's speed.

Next, create a new pygame display surface—the canvas on which in-game items will be drawn—and add the following two lines to it:

```python
playSurface = pygame.display.set_mode((640, 480))
```
The software will now be given some colors to work with. Although this step isn't technically necessary, it does help you save time typing.

To alter the color of a specific item, instead of running the pygame, you might simply use the red Colour variable.

Color instruction and the three color values for red, green, and blue should be remembered. Type the following lines to define the colors for this example program: red

```python
pygame is the color. black color (255, 0, 0)
```

```python
pygame is the color.
```

```python
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```python
```

```python
pygame is the color. the color (150, 150, 150)
```
The next few lines prepare several of the game's variables for use by initializing them. The use of the US English spelling of "color" is an artifact of Pygame's development in the United States; if you write "color," it will not be recognized. Initializing these variables is crucial because if they are left blank when the game starts, Python will have no idea what to do. For now, don't worry about what each variable does; simply input the following lines (making sure to use all of the commas and square brackets correctly):

\[100,100\] snakePosition

\[[100,100],[80,100],[60,100]\] snakeSegments

\[300,300\] raspberryPosition raspberry spawned = 1 change in direction = 'right'

Direction is same to direction.

Three of the variables are set to a list of comma-separated values: snakePosition, snakeSegments, and raspberryPosition. Python creates the variables as lists, which are a collection of various values kept in a single variable name. You'll learn how to access individual values contained in a list later.
The next step is to create a new function, which is a piece of Python code that can be used later in the program. Functions can help you minimize code repetition and make your application easier to understand. If you have a set of instructions that you need at many points in the same program, creating a function with def means you only have to type them once—and you only have to change them once if you change the program later. To define the gameOver function, type the following lines:

gameOver(): def gameOver(): def gameOver(): class game pygame.font.Font ('freesansbold.ttf', 72) gameOverFont = pygame.font.Font ('freesansbold.ttf', 72) gameOverFont.render = gameOverSurf (GreyColour, True, 'Game Over') gameOverRect = gameOverSurf.get rect() gameOverRect.midtop = gameOverSurf.get rect() gameOverSurf.get rect() (320, 10) playSurface.blit is a.blit file that allows you to play with (gameOverSurf, gameOverRect) time.sleep pygame.display.flip() (5) sys.exit pygame.quit() ()

A function's code, like that of loops, should be indented. Every line following the def instruction should begin with four spaces; if you're using IDLE, these spaces are
automatically inserted; if you're using a text editor, you must manually enter the spaces. You can cease indenting after the function's final line—sys.exit().

The gameOver method performs a simple task by combining a number of pygame commands: Using a large font, write the words Game Over on the screen, stop for five seconds, and then exit both pygame and Python. It may seem odd to set up instructions for quitting the game before it ever starts, but functions should always be specified before they are invoked. Python will not execute these instructions unless the newly constructed gameOver instruction tells it to.

Now that the program's starting has been completed, it's time to move on to the major section. This happens in an endless loop, which is a while loop that never ends. This allows the game to continue until the player dies by colliding with a wall or eating his or her own tail. Begin with the following line in the main loop: while it is correct:

Python tests to see if True is true in the absence of anything to evaluate. Because this is always the case, the loop will keep running indefinitely—or at least until you tell Python to stop by executing the gameOver method.
Continue with the lines below, paying close attention to the indentation levels: in pygame.event.get() for event: pygame.quit if event.type == QUIT ()

sys.exit()

elif event.type == KEYDOWN: The first line, which comes after the while loop, should be indented four spaces—but it's its own loop, employing a for instruction to check for pygame events like key pushes. As a result, the line under for needs to be indented four spaces more, for a total of eight spaces—but that line is employing an if command to verify whether the user has touched a key. As a result, the following line, pygame.quit(), is indented four spaces farther, for a total of 12 spaces. This logical development of indentation tells Python where each loop block begins and ends, which is critical because the program will fail if the wrong number of spaces are used. This is why creating Python programs with a development environment like IDLE, which attempts to automatically indent code where necessary, can be easier than using a plain text editor. An if statement instructs Python to examine whether a given evaluation is correct. If pygame reports a QUIT message, the first check, if event.type == QUIT, directs Python to
run the indented code following (which happens when the user presses the Escape key or when the user clicks the X button at the top-right to close the window). The two lines below that should be recognizable from the gameOver function: they close down and exit pygame and Python.

If checks are extended by the line beginning with elif. An elif instruction, which stands for else if, is assessed when a preceding if instruction is judged to be false. The elif instruction is used in this example to see if pygame is reporting a KEYDOWN event, which is returned when a key on the keyboard is pressed. Code to be executed when an elif is true should be indented by four spaces plus the indentation of the elif instruction itself, just like the if instruction. To give the elif instruction something to do when the user pushes a key, type the following lines: if event.key is K RIGHT or ord('d'):

'right' for changeDirection if event.key equals K LEFT or event.key equals ord('a'):

if event.key == K UP or event.key == ord('w'):
    changeDirection = 'up' if event.key == K DOWN or event.key == ord ('s'):
    changeDirection = 'down' if event.key == K DOWN or event.key == ord ('s'):
changeDirection = 'down' if event.key == K DOWN or
event.key                   pygame.event.post   pygame.event.post
pygame.event.post           pygame.event.post
(pygame.event.Event(QUIT))

These instructions adjust the value of the changeDirection variable, which controls the direction in which the player's snake moves during the game. When you use or with an if statement, you can perform several evaluations. In this situation, the player has two options for moving the snake: the cursor keys or the W, A, S, and D keys can be used to move the snake up, left, down, or right. The snake goes to the right until a key is pushed, according to the value set for direction at the start of the program.

The variable direction is used in conjunction with changeDirection to determine whether the user's instruction is valid. The snake should not be allowed to turn back on itself; if it does, the snake will die and the game will be ended. To prevent this from happening, the player's requested direction (stored in changeDirection) is compared to the snake's current direction (stored in direction). If they are in opposite directions, the snake ignores the instruction and continues in the same
direction. To set up the comparisons, type the following lines, alternating between four and eight spaces indentations:

```
if changeDirection == 'right' and not direction == 'left'
direction = changeDirection
```

If there is a shift in direction, Direction Equals 'up,' not 'down,' since direction = change. If there is a shift in direction, direction = changeDirection;

direction = changeDirection;
direction = changeDirection;
direction = changeDirection;
direction = changeDirection;
direction = changeDirection;
```
```

The snake, which shows on the screen as a series of blocks, can be moved after the user's input has been double-checked for accuracy. The snake moves a distance proportional to the size of one of its blocky segments throughout each turn. You may instruct pygame to move the snake a single segment in any direction because each segment is 20 pixels long. Please enter the following code:

If the direction is 'right,' snakePosition[0] += 20; if the direction is 'left,' snakePosition[0] += 20. -= 20 if up is the direction:

```
if direction == 'down' 20 +=
```
The operators += and -= change the value of a variable by a specified amount: += sets the variable to its previous value plus the new value, while -= sets the variable to its previous value minus the new value. SnakePosition[0] += 20 is a shorthand way of writing snakePosition[0] = snakePosition[0] + 20, for example. The position in the list influenced by the snakePosition variable is indicated by the number in square brackets after the variable name. The snakePosition list's first value stores the snake's x-axis position, while the second value stores the snake's y-axis position, with 0,0 representing the upper-left corner. Python starts counting at zero, so snakePosition[0] controls the x-axis and snakePosition[1] controls the y-axis. If the list were larger, increasing the number would effect more entries: [2], [3], and so on.

Although the snakePosition list is always two values long, the snakeSegments list, which was established at the start of the program, is not. The location of the snake's body, behind the head, is saved in this list. This list grows in size as the snake consumes raspberries and grows longer, adding to the game's difficulty: as the player proceeds, it becomes more difficult to avoid hitting the snake's body with the head. If the snake's head collides with its body, it
dies and the game is finished. To make the snake's body grow, type the following line:

snakeSegments.

insert(0,list(snakePosition))

The insert instruction is used to add a new value to the snakeSegments list: the snake's current position. When Python reaches this line, it lengthens the snake's body by one segment and places that segment where the snake's head is now located. It will appear to the gamer that the snake is growing. This should only happen when the snake eats a raspberry; otherwise, the snake will continue to grow. Write the lines below: if snakePosition[0] = raspberryPosition[0] and snakePosition[1] = raspberryPosition[1] If spawned = 0 is true, then snakeSegments.pop is true ()

The first instruction compares the X and Y coordinates of the snake's head to the X and Y coordinates of the raspberry, which is the player's target. The raspberry is assumed to have been eaten by the snake if the values match, and the raspberrySpawned variable is set to 0. Python knows what to do if the raspberry hasn't been eaten
by using the else instruction: From the snakeSegments list, select the "oldest" option.

The pop instruction is straightforward but effective. It returns the list's last value while also removing it, resulting in a one-item reduction in the list's length. In the example of the snakeSegment list, it instructs Python to remove the snake's body segment furthest from the head. It will appear to the player that the entire snake has moved without growing, but in fact it has grown at one end and shrunk at the other. The pop instruction is only executed when a raspberry has not been eaten, thanks to the else expression. When a raspberry is consumed, the last entry in the list isn't removed, therefore the snake expands by one segment.

It's possible that the player has eaten a raspberry at this point in the game. Because a game with only one raspberry is uninteresting, type the lines below to add a fresh raspberry to the playing area if the player has eaten the previous raspberry:

if raspberrySpawned is equal to 0:
random.randrange(1,32) x = random.randrange(1,32) y = random.randrange(1,32) (1,24)

[x*20,y*20] raspberryPosition

1 if raspberrySpawned = 1 if raspberrySpawned = 1 if

This line of code examines if the raspberry has now been eaten by checking if the raspberrySpawned variables is set to 0, and if it is, the code chooses a random place on the playing area using the spontaneous module you loaded at the beginning of the program.

The size of a snake's segment—20 pixels wide and 20 pixels tall—is then multiplied by this location to offer Python a spot on the playing field to place the new raspberry.

It's critical that the raspberry's placement be chosen at random.

This keeps the game exciting by preventing the player from figuring out where the raspberry will emerge next.

Lastly, the raspberry Spawned variables is reset to 1, ensuring that only one raspberry is present on the playing field at any one time.
You now have the code to make the snake move and grow, as well as eat and create raspberries—a process known in gaming as respawning.

Nothing, however, is drawn to the screen.

Write the lines below:

For location in the snake, use playSurface.fill(blackColour).

```
pygame.draw.rect(pygame.draw.rect(pygame.draw.rect(pygame.
(playSurface,white

Color,Rect(position[0], position[1], 20, 20)))

pygame.draw.rect(playSurface,redColour,Rect
pygame.draw.rect(playSurface,redColour,Rect
pygame.draw.rect(playSurface,redColour,Rect
(raspberry raspberry, position [0]

(isplay.flip()
```
These lines instruct pygame to draw the raspberry in red, fill in the base of the playing surface with black, and then draw the snake's body segments in white.

The last line, `pygame.display.flip()`, instructs pygame to update the screen; items will be invisible to the player if this command is not given.

Ensure to use `pygame.display.flip()` after you've finished drawing things onto the screen so the user can see the change.

The snake is currently incapable of dying.

A game in which the user can never die would quickly become tedious, therefore add the following lines to create different death scenarios for the snake:

```python
gameOver() if snakePosition[0] > 620 or snakePosition[0] > 0; gameOver() if snakePosition[1] > 460 or snakePosition[1] > 0; ()
```

The first if statement determines if the snake has crossed the playing surface horizontally, while the second if statement determines whether the snake has crossed it vertically.
In either scenario, the snake is doomed; the gameOver function, which was defined earlier in the program, is used to print a message to the screen and end the game.

If the snake's head collides with any part of its body, it should die, therefore add the following lines:

```python
```

The for statement compares the current position of the snake's head to the locations of each of the snake segments, starting with the second list entry and ending with the last list entry.

It's critical to begin the comparison using `snakeSegments[1:]` rather than the first entry.

The first entry is always the position of the head, therefore starting the analysis there would result in the snake dying instantly as soon as the game started.

Finally, utilizing the fpsClock variable to regulate the update pace is all that is required to finish the game.
The game would run too fast to play without the variable you created at the start of the program.

To complete the program, type the following line:

`fpsClock.tick(20)`

You can raise this number (to update the game at a higher number of frames per second) if you think the game is too easy or too sluggish, or lower it if you think the game is too hard or too quick.

Save the program as raspberrysnake.py and run it from the terminal by entering `python raspberrysnake.py` or from IDLE's Run Module option in the Run menu.

Make sure you're ready since the game will begin as soon as it's loaded (see Figure 11-6).

Although you can save time by obtaining the source code from the internet, inputting the code by hand ensures that you comprehend what each section performs.

In addition to the capabilities you've used in Raspberry Snake, pygame has a number of additional features, such as audio playback, sprite handling for improved graphics, and mouse control.

The official website, www.pygame.org/wiki/tutorials, is the ideal place to learn about pygame's more complex functionalities, where you can install tutorials and example programs to get a feel for how things work.

Python and Networking (Example 4)

You've learned how to develop independent programs using Python so far, but the code can also be used to create programs that interact with the outside world via a computer's network connection.
Tom Hudson's following example, a tool for monitoring users connected to an Internet Relay Chat (IRC) channel, provides a small taste of these possibilities.

Create a new project in IDLE or a text editor as usual, and enter the shebang line as well as a description outlining the program's purpose: 

```
#!/usr/bin/env python # IRC Channel Checker, created by Tom Hudson for the Raspberry Pi User Guide.
```

Then, with the following line, import the program's required modules—sys, socket, and time: 

```
sys, socket, and time are imported.
```

In the Raspberry Snake program, you used the sys and time modules previously, but you haven't yet utilized socket.

The socket module allows Python programs to open, close, read from, and publish to network sockets, providing them with basic networking capabilities.

This example's able to connect to a remote IRC server is thanks to the socket module.

For this program to work, some constants are required.
Constants are similar to variables in that they can be assigned values.

However, unlike variables, the value of a constant should not change.

To make it easier to tell the difference between a constant and a variable, use all-capital letters for their names. This way, you can tell whether a line of code is using a constant or a variable at a look.

It's worth noting, though, that Python sees no difference between upper- and lower-case typography; it's simply there to help programmers.

In the program, write the following two lines:

```
RPL ENDOFNAME = '366' RPL NAMREPLY = '353'
```

These are IRC status codes that the server provides to signal when specific operations are complete.

These codes are used by the application to detect when the required list of names has been received from the IRC server.

Next, enter the following lines to set up the server connection variables:

```
irc = 'host': 'chat.freenode.net', 'port':
```
Python is told to generate a dict data type in the first line.

This permits several parameters to be stored in a single master variable—in this case, irc—and is short for dictionary.

Later in the software, these individual variables can be recalled.

Although you could create this program without utilizing dicts to hold variables, it would be far more difficult to understand.

The dict starts with the opening curly brace and concludes on the last line with the closing curly brace.

Set the host variable to the IRC server's fully qualified domain name (FQDN), which the program will connect to.

Chat.freenode.net is used in this example, but you can change the domain name here if you want to customize the software to use a different server.
The port variable is a variable that can be changed. Specifies the network port on which IRC is running, which is commonly 6667.

The channel parameter tells Python which channels to join in order to watch the users, and the names parameter tells Python which people to monitor.

The interval setting determines how long the software waits to update the user list, in seconds.

Type the following lines to create a second dict to store the user-specific variables:
```
user = 'nick': 'botnick,' username: 'botuser,' hostname: 'localhost,' servername: 'localhost,' realname: 'Raspberry Pi Names Bot'
```

All of these variables are put in a dict called user, just like in irc, to make it apparent which variables belong to which part.

Set the nick variable to the IRC pseudonym that the program will use.

If you're going to connect to the IRC server at the same time, add -bot to the end of your name to indicate that the users is a program rather than a real person.
Fill in the real name variable with an informative message about who the bot belongs to, and do the same with username.

You can leave the hostname and servername variables at localhost or change them to match your Internet address.

The socket module necessitates the creation of a socket object by the user.

This object connects the remainder of the application to the internet.

Type the following line to create the socket object:

Next, tell the program to try connect to the IRC server given in the variables at the start of the program with s = socket.socket(socket.AF_INET, socket.SOCK_STREAM).

Write the lines below:

'Connecting to (host)s: (port)' print ' (port)
irc s...' percent irc s...' percent irc s...
s.connect((irc['host'], irc['port'])) is a good place to start.
except for the socketerror: 'Error connecting to IRC server'
print irc percent (hosts): percent (ports) sys.exit(1)
For error handling, the try and except instructions are included in this code.

The software will output an error message and gracefully depart if the system fails to access the server example, because the Pi isn’t connected to the Internet or because the server is unavailable for maintenance.

The `s.connect` line instructs the socket module to attempt to connect to an IRC server using the hostname and port variables from the `irc` dict.

The program has linked to the IRC server if it does not exit from the exception.

However, before you can retrieve a list of names on a channel, you must first identify yourself to the server and transmit some commands using the socket module’s `send` function.

In the program, type the following lines:

```python
s.send('NICK percent (nick)srn' percent user) s.send('NICK percent (nick)srn' percent user) s.send('NICK percent (nick)srn' percent user) s.send('USER percent (username)s percent (hostname)s percent (servername)s: percent (realname)srn' percent
```
user) s.send('USER percent (username)s percent (hostname)s percent (servername)s: percent (realname)srn'
percent user) s.send('USER percent (username)s
s.send('JOIN percent (channel)srn' percent irc)
s.send('JOIN percent (channel)srn' percent irc)
s.send('JOIN percent (channel)srn' percent irc)
s.send('NAMES percent (channel)srn' percent irc)
s.send('NAMES percent (channel)srn' percent irc)
s.send('NAMES percent (channel)srn' percent irc)

The send function is very identical to the print function, except it delivers the output across the network connection rather than printing to the conventional output (typically the terminal window or console).

In this case, the program is simulating a user pressing the Enter key by sending thread of text to the IRC server, followed by carriage return (r) and newline (r) characters, telling it to register the program using the nickname stored in the nick variable and the user details stored in the username, hostname, servername, and realname variables.
The software then sends the command to join the channel provided in the channel variable, followed by the command to get a list of users in that channel.

Although this example is specific to IRC, the same basic approach may be applied to any network service—with a few tweaks, this program might be used to show files on an FTP server or unread emails on a POP3 server.

It's a little more difficult to get data from the socket.

To begin, establish an empty string variable that will function as the receive buffer, storing data received from the server until it can be processed.

Type the following line to initialize the buffer: read buffer = "After the equals sign, there are two single quotations, not one double quote.

Next, type the following code to create an empty list that will be used to hold user names: names = []

The list type of data is the same as the one you used in the Raspberry Snake game to store the locations.

It can store several values, unlike a typical variable, in this case the names of users in the IRC channel.
The program then enters an indefinite loop, in which it repeatedly requests the server for pseudonyms and prints them to the screen.

To begin the loop, type the following:

```
read buffer += s.recv while True (1024) While True: directs
the socket module to accept (up to) 1024 bytes (1K) of data
from the IRC server and store it in the read buffer variable
on the first line of the loop.

The received data will be added to whatever already in the
buffer because the += operator is used instead of merely =.

The number 1024 bytes was chosen at random.

The next step is to use the following program lines to split
the buffer into individual lines of text: read buffer.split('rn')
lines = read buffer.split('rn')
lines.pop(); read buffer = lines.pop();

The first line uses the split function to locate end-of-line
characters, which are represented by rn, and sets the lines
variable to entire lines of text from the receive buffer.
Because these characters only appear at the end of a line, you can be sure that the lines in the buffer contain only full-line response from the server.

Because replies from the server are read in 1K chunks, the end of the read buffer is likely to contain a fraction of a line at any one time. The pop command in the second line ensures that only entire lines are removed from the read buffer.

When this happens, the fraction is saved in the buffer, ready to receive the rest of the line the next time the loop runs and the server sends the next 1K chunk.

The lines variable now holds a list of the server's full responses—full lines—received at this point.

To process these lines and find the names of channel participants, type the following: for a line between lines:


if response code == RPL NAMREPLY: if response code == RPL NAMREPLY: if response code == RPL NAMREPLY: if response code == response [3].split(':') = names list[1]
This loops through all of the lines in the lines variable, looking for the server's numerical IRC answer code.

Despite the fact that there are many distinct response codes, this programme is only interested in the two that are defined as constants at the start:

353, which indicates the start of a list of names, and 366, which indicates the end of the list.

If the first of these responses is found, the if statement utilizes the split function to obtain the names and add them to the names list.

The names list now contains all of the names that the server returned in response to the program's query.

However, because the list is incomplete until the 366 response, which signifies the conclusion of the user names, is received, this may not be all of the names.

That is why the last line—`names += names list.split(' ')`—appends the newly received names to the current list rather than blanking it out entirely: the program is likely to have
gotten only a subset of the complete member list each time that section of the code executes.

Enter the following lines, starting with eight spaces, to inform Python what to do when the whole list is received:

If response code is RPL ENDOFNAMES, then: #

Names should be displayed.

percent irc for name in names: print 'rnUsers in percent (channel)s:'

[] print names = [] print names = [] print names = [] print names

This instructs Python to print the now-complete list of names to standard output after receiving the 366 answer, before blurting the names list again.

The last line, names = [], is crucial.

Without it, the loop will add users' names to the list every time it runs, even if they already exist from a previous iteration.

Finally, input the following lines to complete the program:

time.sleep(irc['namesinterval'])
This instructs Python to wait the amount of seconds specified by namesinterval before sending another query for usernames and restarting the loop.

Make sure namesinterval is set to a fair value; if the IRC computer receives too many requests in too short a time, it may disconnect you for flooding.

Save the program as irc userlist.py and run it from the terminal by running python ircuserlist.py or using IDLE's Run Module option in the Run menu.

It may take a bit for the software to connect to the server the first time it runs; once connected, the list of names (see Figure 11-7) should refresh quickly.
Press Ctrl+C to exit the program.

Appendix A of the Raspberry Pi User Guide webpage at www.wiley.com/go/raspberrypiuserguide4 contain a complete copy of the program listing for the IRC user list.

Although you can save time by obtaining the source code from the internet, inputting the code by hand is an excellent approach to ensure that you understood what each section performs.