A Brief Guide To Migrating From Buster To Bullseye

Raspberry Pi Ltd

2022-04-29: githash: ba7441c-clean

Colophon

build-date: 2022-04-29
build-version: githash: ba7441c-clean

Legal Disclaimer Notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI LTD ("RPL") "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL's Standard Terms. RPL's provision of the RESOURCES does not expand or otherwise modify RPL's Standard Terms including but not limited to the disclaimers and warranties expressed in them.

Document version history

Release	Date	Description
1.0	25 November 2021	Initial draft release
1.1	27 April 2022	Copy edit, public release

Scope of document

This document applies to the following Raspberry Pi products:

Pi 0			Pi 02	Pi 1		Pi 2		Pi 3	Pi 4	Pi 400	CM 1	CM 3	CM 4	Pico
0	W	H	W	A	B	A	B	B	All	All	All	All	All	All
*	*	*	*	*	*	*	*	*	*	*	*	*	*	

Introduction

This whitepaper assumes that the Raspberry Pi is running the Raspberry Pi operating system (OS), and is fully up to date with the latest firmware and kernels.

The Raspberry Pi OS is provided by Raspberry Pi Ltd to run on the Raspberry Pi Ltd devices indicated in the previous section. The Raspberry Pi OS is a Linux OS based on the Debian distribution. This distribution is updated approximately every two years, and the Raspberry Pi OS follows these updates, usually a few months after the main Debian release. Debian releases are given codenames; the most recent two at the time of writing have been called Buster and Bullseye. While Raspberry Pi Ltd continually provides incremental updates between major OS releases, at the point of these named release changes some major changes can be made. In addition, Raspberry Pi Ltd does make other changes to the system during these named updates, over and above those provided by Debian.

This document describes some of the major changes made between the Raspberry Pi OS Buster and Bullseye releases, providing examples where procedures may need to be changed. It does not cover all possible differences between the OS versions.

Terminology

Legacy graphics stack: A graphics stack wholly implemented in the VideoCore firmware blob exposed by a Linux framebuffer driver. This is what has been used on the majority of Raspberry Pi Ltd Pi devices since launch, but is gradually being replaced by (F)KMS/DRM.

FKMS (vc4-fkms-v3d): Fake Kernel Mode Setting. While the firmware still controls the low-level hardware (for example the High-Definition Multimedia Interface (HDMI) ports, Display Serial Interface (DSI), etc.), standard Linux libraries are used in the kernel itself. FKMS is used by default in Buster, but is now deprecated in favour of KMS in Bullseye.

KMS (vc4-kms-v3d): The full Kernel Mode Setting driver. Controls the entire display process, including talking to the hardware directly with no firmware interaction. Used by default on Bullseye.

DRM: Direct rendering manager, a subsystem of the Linux kernel used to communicate with graphics processing units (GPUs). Used in partnership with FKMS and KMS.

Config.txt

There are many references in this whitepaper to the `config.txt` file, which is found in the `boot` folder in a Raspberry Pi OS installation. While some usage is described below, refer to the official documentation for more detailed descriptions of the commands.

To edit the file:

```
sudo nano /boot/config.txt
```

NOTE

In this document, `nano` is used as the text editor since it is installed in Raspberry Pi OS by default. Feel free to substitute your text editor of choice.

Kernel command line

The kernel command line is stored in the file `/boot/cmdline.txt`; this document will occasionally suggest changes that need to be made to this file. When editing it, ensure you do not add any carriage returns — this file must consist of only a single line.

To edit the file:

```
sudo nano /boot/cmdline.txt
```

The main differences between Buster and Bullseye

As with all Debian releases, a large number of third-party applications will have been updated to more recent versions. It should be noted that although these application are more recent, they will not necessarily be the very latest versions. Debian is very conservative when it comes to software releases, and stability is much more important than having the very latest features. Therefore, Debian tends to lag behind the very latest releases in order to ensure a reliable system. This document will not cover how to get more recent versions of software than those installed by `apt` in Bullseye.

Some major architectural changes have been made to the Raspberry Pi OS with the Bullseye release. The main ones are listed here, and are described in more detail in the appropriate subsections:

- Move to KMS from legacy graphics or FKMS
- Move from the legacy closed source camera stack to the open source libcamera system
- Deprecation of Multi-Media Abstraction Layer (MMAL) and OpenMAX, and move to Video4Linux (V4L2) for codec and video pipeline support
- Desktop composition moved from Openbox to Mutter (on devices with 2GB or more of random access memory)

Many of these changes are intended to provide a more open system, moving away from code based on closed source firmware to open source application programming interfaces (APIs), thereby allowing users better access to the underlying hardware that was previously only usable via proprietary APIs. This means that instead of many parts of the hardware only being controlled from the VideoCore processor, and therefore closed source, these hardware blocks are now controllable from the Arm cores via standard Linux APIs. Another very important side effect of these changes is that all the new libraries are available when running a 64-bit OS, whereas many of the older ones will simply not work in that environment.

IMPORTANT

In some cases it is possible to install and use older APIs on Bullseye, and this is described where it is possible. However, there is no guarantee that any legacy APIs will be able to be installed in the future, so it is advisable to start migration to the newer APIs as soon as possible.

In summary, the changes can be described as follows:

Previous	New	Comments
Buster	Bullseye	Many upstream (from Debian) packages have been updated.
Legacy/FKMS	KMS	This is a major change to the way the display hardware is controlled.
Firmware camera stack	libcamera	Picamera Python bindings not yet supported.
Openbox	Mutter	Only on 2GB devices and above.
MMAL/OpenMAX	V4L2	

KMS

Kernel Mode Setting is a standard Linux API for controlling graphics output. Combined with the DRM library it controls all output to display devices, such as HDMI and composite monitors, LCD panels, etc. In Buster the FKMS graphics driver was used; this provided a KMS API, but instead of controlling the hardware directly, it was simply a shim over the underlying firmware-based graphics drivers. With KMS the driver talks directly to the hardware, and the firmware is not involved.

The FKMS system introduced some features that are continued with Bullseye, such as a standard way of setting display orientation. In addition, moving to an entirely kernel-based system with no firmware involvement means that a number of new features are exposed:

- The ability to write your own DSI (LCD panel interface) drivers
- Support for HDMI hotplugging
- No binary blobs of closed source graphics code
- Multiple displays fully integrated, including mixing HDMI, composite, DSI, and Display Parallel Interface (DPI), with up to three displays possible in some combinations
- Standard API to control the display modes

Display resolution/mode

Before Bullseye

Various custom `config.txt` options were available for setting specific HDMI and Digital Visual Interface (DVI) modes and resolution (`hdmi_mode`, `hdmi_group`, and `hdmi_timings`). See the `config.txt` documentation for full details.

Bullseye onwards

In Bullseye, mode and resolution selection is done via the kernel command line.

The following example tries to set a 1024×768 mode at 60Hz on HDMI port 1:

```
video=HDMI-1:1024x768@60
```

The display referenced can be one of `HDMI-1`, `HDMI-2`, `DSI-1`, `Composite-1`, `DSI-2`, `DPI-1` or `LVDS-1`; the options available will depend on the particular hardware configuration.

Adding a `D` to the end of the entry will force that display to be enabled and to use digital output, e.g.

```
video=HDMI-A-1:1024x768@60D
```

NOTE

The command line settings will select a mode of the requested resolution/refresh rate if it is defined in the Extended Display Identification Data (EDID)/panel configuration, otherwise it will create a new mode using those parameters based on the standard Coordinated Video Timing (CVT) timings algorithm.

There is more information on setting display modes and the options that are available in the `modedb` kernel documentation.

Display orientation

While the `xrandr` command line application provides the ability to rotate and flip displays in KMS, FKMS and legacy graphics systems require entries in the `config.txt` file to achieve certain features, for example controls such as `display_lcd_rotate` and `display_hdmi_rotate`. These controls are not used in Bullseye/KMS, and will have no effect. Equivalent functionality is available when using the desktop by using the `arandr` command from the command line, or the screen configuration utility from the main menu. If you are using a device in console mode only, then you need to add a specific entry to the kernel command line to achieve the required orientation.

Before Bullseye

Use the `display_lcd_rotate` and `display_hdmi_rotate` entries in `config.txt`; `display_lcd_rotate` will also set up any touchscreen appropriately.

Bullseye onwards

If using the desktop, use the screen configuration utility to make changes.

If you are only using a console mode, you need to update the kernel command line. For example, to rotate a DSI display add the following to the kernel command line:

```
video=DSI-1:800x480@60,rotate=180
```

Displays can be manually rearranged with:

```
xrandr --output HDMI-1 --left-of HDMI-2
```

where the location can also be `--right-of`, `--below`, or `--above`. The display can be one of `HDMI-1`, `HDMI-2`, `DSI-1`, `Composite-1`, or potentially `DSI-2`, `DPI-1`, or `LVDS-1` depending on the hardware attached.

There are also helpers to map touchscreen input data to the appropriate display. This is useful when you have multiple displays attached and need to direct touchscreen input to a specific display.

The following command line example maps the touchscreen input to DSI-1 (note that this is done automatically by `xrandr` if it finds the touchscreen installed):

```
xinput --map-to-output "generic ft5x06 (79)" DSI-1
```

You can change the touchscreen orientation as follows:

```
xinput --map-to-output "generic ft5x06 (79)" DSI-1
```

Overscan

Before Bullseye

In `config.txt`, use the `overscan_left`, `overscan_right`, `overscan_top`, and `overscan_bottom` entries to specify the appropriate margins.

Bullseye onwards

KMS under Bullseye handles overscan in a very different way to Buster. The margins are set on the kernel command line, as in the following example:

```
video=HDMI-
1:720x576@50i,margin_left=10,margin_right=10,margin_top=15,margin_bottom=15
```

You can specify any of the standard output devices instead of `HDMI-1`, or have multiple entries on the command line for multiple devices.

NOTE

Even when using Bullseye/KMS, the firmware will convert any `config.txt` `overscan_` entries into kernel command line `margin` entries automatically. You should only need to manually set these, as shown above, when requiring different margins on multiple displays.

You can use the Raspberry Pi Preferences application when running the desktop to set overscan/underscan parameters, and this will set the required `config.txt` entries.

Using EDIDs

EDIDs are chunks of data recovered from attached display devices that support this method of probing (HDMI, DVI, or VGA). They define all the available graphics and audio modes available on the device. On occasion, it may be useful to know the content of a device's EDID, or even replace it with a custom EDID.

Before Bullseye

For the Raspberry Pi OS prior to Bullseye, EDID data could be recovered from the attached device using the `tvservice` command. To create a file called `edid.bin` in the `boot` folder (they must be here or in a subfolder of `boot`) containing the EDID data, use the following:

```
sudo tvservice -d /boot/edid.bin
```

In order to use a custom EDID you add a `config.txt` command to tell the firmware to do so, and where to find it:

```
hdmi_edid_file=1
hdmi_edid_filename=edid.bin
```

Bullseye onwards

In Bullseye the `tvservice` application is deprecated, and there is a new (more standard) mechanism to recover the EDID data by exporting it from the kernel via a `sysfs` interface. In the example below, we copy this to the location `/lib/firmware`:

```
sudo cp /sys/devices/platform/gpu/drm/card1/card1-HDMI-A-1/edid
/lib/firmware/edid.bin
```

Sometimes, instead of `card1-HDMI-A-1` it may be `card0-HDMI-A-1`.

To use that EDID, or any custom EDID, we need to alter the kernel command line, which can be found in the `boot` folder. Note that the EDID file *must* be in `/lib/firmware`. Add the following to the end of the kernel command line:

```
drm.edid_firmware=edid.bin
```

`drm.edid_firmware=` also supports a comma-separated list for attaching to a specific connector, which allows you to assign different EDIDs to multiple devices, e.g.

```
drm.edid_firmware=HDMI-1:edid1.bin,HDMI-2:edid2.bin"
```

Omitting the connector name will apply the EDID to any connector that does not explicitly match an entry.

New display timings

Although EDID support should cover most display situations, specific mode timings occasionally need to be set for custom displays and similar.

Video Electronic Standards Association (VESA) CVT is a standard timing algorithm (`man cvt` for details) which converts resolution and frequency to a set of timing numbers. However, not all displays want CVT timings, so there is also an app `gtf` for the VESA Generalized Timing Formula (see `man gtf`), or it is possible to specify any particular timing required by the display.

Before Bullseye

Display timings could be set using the `cvt_timings` and `hdmi_timings` options in `config.txt`.

Bullseye onwards

The `cvt_timings` and `hdmi_timings` options are ignored in Bullseye. The `xrandr` Linux command can be used as an alternative, although this is done later in the boot cycle, once Linux is up and running.

The process is: register a new display mode with the required timing, add this mode to the display to which you want it to apply, and select it. The display timings may come from the supplier of the display device, or you can generate timings for

specific modes using the `cvt` command. For example, for a 1200×700 display at 60Hz, you get the following modeline:

```
$ cvt 1200 700 60
# 1200x700 59.82 Hz (CVT) hsync: 43.49 kHz; pclk: 67.50 MHz
Modeline "1200x700_60.00"   67.50  1200 1256 1376 1552  700 703 713 727 -hsync
+vsync
```

You can now add this mode to `HDMI-1` as follows:

```
xrandr --newmode "1200x700_60.00"   67.50  1200 1256 1376 1552  700 703 713 727
-hsync +vsync
xrandr --addmode HDMI-1 1200x700_60.00
xrandr --output HDMI-1 -mode 1200x700_60.00
```

These settings will be lost on a reboot, so you will need to add these commands to a startup file or similar so they are executed each time the X Windows system starts up, for example in a desktop autostart file such as `~/.config/autostart/.desktop`.

Setting display properties

KMS properties are one way of setting display options that may have been possible prior to Bullseye with `config.txt` entries.

You can list and set the available properties using `xrandr`:

```
# List properties for displays
xrandr --prop
# Set properties on specified display
xrandr --output HDMI-1 --set <prop_name> <value>
```

NOTE

When changing margins via properties, you will need to switch away from the desktop display to a console (Ctrl-Alt-F2) then back to the desktop (usually Ctrl-Alt-F7) for the changes to take effect. This is because the margin values are only used when the desktop display is initially created, so it needs to be destroyed and recreated for the changes to take effect.

Turning off the display

Prior to Bullseye the `tvservice` command allowed you to turn off the display output completely. In Bullseye, this is now done with the `xrandr` command. To turn HDMI-1 off and on, use the following:

```
# Turn HDMI off with
xrandr --output HDMI-1 --off
# Turn it back on with
xrandr --output HDMI-1 --preferred
```

You can also use other display identifiers.

DSI display

A huge advantage of using KMS is the ability to use DSI displays other than the Raspberry Pi Ltd device, as long as you have an appropriate driver. However, the move to KMS also means changes are needed to support the Raspberry Pi Ltd display, to enable it, and to change orientation.

If `display_auto_detect=1` is set in `config.txt`, then the `vc4-kms-dsi-7inch` overlay will be automatically selected if the DSI screen was detected by the firmware *and* `vc4-kms-v3d` is also detected. Note that `display_auto_detect=1` is set by default in Bullseye.

Autodetection only works for official Raspberry Pi Ltd displays, so if you have a different display, or cannot use `display_auto_detect` for some reason, you need to enable DSI displays via a device tree overlay. To do so, add the following line in `config.txt` (this one is for the official display; you will need to use the correct overlay for your display):

```
dtoverlay=vc4-kms-dsi-7inch
```

There is a whitepaper available that goes into a lot more detail on the use of the DSI display interface under KMS.

DPI displays

In Buster, DPI displays are set up using a simple entry in `config.txt`. In Bullseye, you need to use device tree entries, and possibly edit the kernel code to define timings.

There are already two DPI panel overlays in the kernel that can be used as examples when defining overlays for other devices:

- Adafruit Kippah display
- Innolux display

While the correct and official way of defining a DPI display's timings is to add them to `panel-simple.c` and use the `panel-dpi` compatible string, Raspberry Pi Ltd provides a modified `panel-dpi` driver where the settings can be specified as device tree parameters. For example:

```
dtoverlay=vc4-kms-v3d
dtoverlay=vc4-kms-dpi-generic,hactive=480,hfp=26,hsync=16,hbp=10
dtparam=vactive=640,vfp=25,vsync=10,vbp=16
dtparam=clock-frequency=32000000,rgb666-padhi
```

This overlay should provide similar functionality to the `dpi-timings` `config.txt` entry.

WARNING

Device tree lines must be less than 80 characters in length, hence the example above splits the timings over multiple entries.

The overlay has the following documentation:

```
Name:   vc4-kms-dpi-generic
Info:   Enable a generic DPI display under KMS. Default timings are for the
```

```
        Adafruit Kippah with 800x480 panel and RGB666 (GPIOs 0-21).
        Requires vc4-kms-v3d to be loaded.
Load:   dtoverlay=vc4-kms-dpi-generic,<param>=<val>
Params: clock-frequency         Display clock frequency (Hz)
        hactive                 Horizontal active pixels
        hfp                     Horizontal front porch
        hsync                   Horizontal sync pulse width
        hbp                     Horizontal back porch
        vactive                 Vertical active lines
        vfp                     Vertical front porch
        vsync                   Vertical sync pulse width
        vbp                     Vertical back porch
        hsync-invert            Horizontal sync active low
        vsync-invert            Vertical sync active low
        de-invert               Data Enable active low
        pixclk-invert           Negative edge pixel clock
        width-mm                Define the screen width in mm
        height-mm               Define the screen height in mm
        rgb565                  Change to RGB565 output on GPIOs 0-19
        rgb666-padhi            Change to RGB666 output on GPIOs 0-9, 12-17, and
                                20-25
        rgb888                  Change to RGB888 output on GPIOs 0-27
        bus-format              Override the bus format for a MEDIA_BUS_FMT_*
                                value. NB also overridden by rgbXXX overrides.
        backlight-gpio          Defines a GPIO to be used for backlight control
                                (default of none).
```

While the process of enabling DPI displays is more complex in Bullseye, you do get the benefits of integration with any other attached displays on the desktop, along with the orientation options described previously. The process once again moves away from Raspberry Pi custom settings to more standard Linux APIs. It is expected that panel suppliers will do much of the hard work of setting up timings in the kernel code, and provide overlays to enable those settings, so that the end user will simply need to add a `dtoverlay` entry in `config.txt` to enable the panel.

There is a whitepaper available that goes into a lot more detail on the use of the DPI display interface under KMS, including the use of panels that require extra initialisation steps.

Unsupported display `config.txt` commands in Bullseye

A number of display-related `config.txt` entries, over and above those described previously, no longer have any effect in Bullseye due to the use of KMS, which replaces or removes all of these features:

- cec_osd_name
- config_hdmi_boost
- disable_touchscreen
- display_default_lcd
- dpi_group
- dpi_mode
- dpi_output_format
- hdmi_force_edid_3d
- hdmi_force_mode
- hdmi_group
- hdmi_ignore_cec
- hdmi_ignore_cec_init
- hdmi_ignore_edid
- hdmi_ignore_edid_audio

- dpi_timings
- edid_content_type
- enable_dpi_lcd
- enable_tvout
- framebuffer_depth
- framebuffer_height
- framebuffer_ignore_alpha
- framebuffer_width
- hdmi_blanking
- hdmi_drive
- hdmi_edid_file
- hdmi_edid_filename
- hdmi_max_pixel_freq
- hdmi_mode
- hdmi_pixel_encoding
- hdmi_safe
- hdmi_timings
- ignore_lcd
- lcd_framerate
- lcd_rotate
- max_framebuffers
- sdtv_aspect
- sdtv_disable_colourburst
- sdtv_mode

libcamera

While the new libcamera API has been available in earlier versions of the Raspberry Pi OS, on Bullseye it is now the default and the older camera applications are no longer installed. Libcamera is a new camera API for Linux, and replaces the custom and closed source camera stack from earlier Raspberry Pi Ltd systems. This means much better access to the camera hardware than was previously available, and adds some very useful new features, but does remove some less used options.

TIP

Why libcamera? libcamera is a new API developed for Linux that is entirely open source. Raspberry Pi Ltd have collaborated with the libcamera developers to make sure it provides almost everything that the previous legacy camera stack does, but with the massive advantage of being entirely open source, easy to tune, and easy to use. Applications written to the libcamera specification will work on any device that supports it, and are not limited to Raspberry Pi Ltd devices.

An important difference between the legacy stack and libcamera is the removal of the `raspistill`, `raspivid`, `raspistillyuv`, and `raspiyuv` applications and their replacement with libcamera-based alternatives. Raspberry Pi Ltd have gone to great lengths to replicate the command line features of these original applications, and for many people this will mean simply changing the name of the application used as per the following table. The most noticeable difference in the applications is the way preview windows are displayed. On Bullseye these now appear in desktop windows (if you are using a desktop) rather than being superimposed over the top of the desktop as in Buster. If you are not using a desktop, the preview uses the whole display.

WARNING

Not all of the short-form versions of the command line options are available in libcamera apps. Use `--help` with the required app to get a list of all the available libcamera commands for that application.

Legacy	libcamera
raspistill	libcamera-still
raspivid	libcamera-vid
raspiraw	libcamera-raw

The `picamera` Python API does not currently work on Bullseye. A new library, `picamera2`, is being developed, but if you need a Python binding you will need to install the legacy camera stack, as covered in a later section.

Finally, libcamera does not yet support stereo cameras. You will need to use the legacy camera stack if stereo cameras are required.

Camera overlays

To make the transition to Bullseye easier, a new `config.txt` flag, `camera_auto_detect`, has been added. If `config.txt` contains `camera_auto_detect=1`, the correct device tree camera overlays will be loaded automatically, according to which cameras have been previously detected by the firmware on startup. Note that `camera_auto_detect=1` is set by default in Bullseye.

NOTE

Autodetection only works for official Raspberry Pi Ltd cameras. Third-party cameras will require the `dtoverlay` entry to be manually entered as described below.

When using libcamera, enabling a specific camera is done via the device tree. This can be done by manually adding a device tree overlay to the `config.txt` file, for example:

```
dtoverlay=imx219
```

Camera detection

On Buster, using the legacy camera stack, you could use `vcgencmd get_camera` to test whether a camera had been detected and enabled. This is not supported by libcamera, and will return 'undetected' even if a camera is attached. An alternative is to check `/dev/video0`: if that is listed as a device node, then the camera has been detected and libcamera is able to use it. Note, though, that `video0` will also be present on FKMS and legacy systems if a camera is detected and the bcm2835-camera module is running, which it does by default. To check for `/dev/video`:

```
ls /dev/video0
```

Another option is to check the installed v4l2 devices:

```
v4l2-ctl --list-devices
```

If a camera is detected, one of the devices reported will be 'unicam (platform:fe801000.csi)' for the new libcamera stack (note that the address will change on Raspberry Pi 0–3), or 'mmal service X.X' for the legacy camera stack.

You can also dump the attached inter-integrated circuit (I2C) devices using `i2cdetect`. If there is a 'UU' against address 0x10 (IMX219), 0x1A (IMX477), or 0x36 (OV5647) then there is a kernel driver active on it, which is the camera driver. Note that other models of camera (not necessarily from Raspberry Pi Ltd) will appear on different addresses.

In the latest version of the libcamera applications, an additional feature has been added to the command line to display the cameras present, `--list-cameras`. For example:

```
libcamera-hello --list-cameras
Available cameras
-----------------
0 : imx477 [4056x3040] (/base/soc/i2c0mux/i2c@1/imx477@1a)
```

Reverting to the legacy camera system

In some cases it may be necessary to revert to the legacy system, and this is possible in Bullseye.

NOTE

Future versions of the Raspberry Pi OS may not support the legacy camera system, so now is the time to start migration to libcamera.

You will need to download and build the legacy raspicam applications using a process similar to this example:

```
cd ~
sudo apt install cmake
mkdir bin
git clone https://github.com/raspberrypi/userland
cd userland
./buildme
cp build/bin/* ~/bin/
```

This will build and copy the raspicam applications to the newly created `bin` folder; they can be run from there.

NOTE

The move to KMS means the preview modes will no longer work, even after the applications have been built. Either use the `-n` option to prevent previews from being displayed, or revert back to the FKMS driver instead by changing `dtoverlay=vc4-kms-v3d` to `dtoverlay=vc4-fkms-v3d` in `config.txt`.

NOTE

The legacy camera applications do not work correctly in a 64-bit environment, and will never do so.

Video pipeline

Prior to Bullseye, all video codecs were handled in firmware via the proprietary MMAL API. This was used in conjunction with the more open OpenMAX API, with MMAL providing an easier way to access the underlying hardware features. However, the MMAL API is only used on Raspberry Pi Ltd devices, so code written on other devices is not immediately compatible. Not only that, but MMAL and OpenMAX are not 64-bit friendly, so do not work very well, if at all, when used on a 64-bit system.

In Bullseye, these APIs have been deprecated in favour of the standard V4L2 Linux API. This means better code compatibility, and immediate compatibility with 64-bit systems.

Code that uses MMAL or OpenMAX will no longer work on Bullseye, so will need to be rewritten to use V4L2. Examples of this can be found in the `libcamera-apps` GitHub repository, where it is used to access the H264 encoder hardware. For example, the MMAL/IL `video_render` component will no longer work, nor will any calls made directly to `dispmanx_` functions.

In addition, OMXPlayer is no longer supported, and for video playback you should use the VLC application. There is no command line compatibility between these applications — see the VLC documentation for details on usage.

When using VLC on the desktop, the output will be displayed in a window unless full screen is selected, unlike OMXPlayer which does not use windows and simply superimposes the video over the desktop.

Desktop window manager

A window manager is responsible for all the window borders, menu display, etc. that you see on the Raspberry Pi OS desktop. In Buster we used the Openbox manager; in Bullseye we have moved to the Mutter manager. The main differences are explained in this blog post, but in short Mutter provides various rendering effects such as drop shadows to make the desktop look a lot more modern.

There are a few things to look out for in the change. As outlined in the blog post, Mutter is only enabled on devices with 2GB of memory or more, so older Raspberry Pi models which were limited to 1GB will still use Openbox. Openbox is also used when the VNC server is enabled as this is not compatible with Mutter.

On the whole, there are no major differences in using the two systems; however, future desktop development will require a compositing windows manager, so now is the time to make sure that Mutter works for you.

Other options

While the recommended approach is to move to the more open and standard systems provided by the Bullseye release, in some case it may not be possible to migrate, perhaps due to application incompatibility. Depending on the use case, it may be possible to disable Bullseye features in order to make legacy systems operate. If this is the case, Raspberry Pi Ltd recommends that this be regarded as a temporary measure, as there is no guarantee that any legacy features will remain in future Raspberry Pi OS releases.

For example, if you do not need the additional features provided by the KMS system, you could revert back to the legacy graphics or FKMS. Edit the `config.txt` file and change the line that says:

```
dtoverlay=vc4-kms-v3d
```

to the following to use FKMS (not recommended since it is unsupported):

```
dtoverlay=vc4-fkms-v3d
```

or to the following to use the legacy graphics:

```
#dtoverlay=vc4-kms-v3d
```

The libcamera section already explains how to use the legacy camera applications.

Remaining on Buster

If you have applications that simply do not work, or work incorrectly, when using Bullseye, it is quite valid to remain on the Buster release. This release is now known as Raspberry Pi OS (Legacy).

The release will follow updates from Debian and the Linux 5.10 kernel, with updates to support product revisions, but not new products. This means that certain features that cannot be supported will be removed (for example, hardware accelerated Chromium). The kernel will receive security updates and hardware support patches, and will remain at 5.10.x.

NOTE

The Legacy release *will not* support new Raspberry Pi Ltd hardware releases, only those devices that are currently in production, e.g. Pi 4, Pi 3B+, Pi 3B, Pi 2, Pi 1, Zero, and Zero 2.

The Legacy distribution can be installed by selecting the Raspberry Pi OS Other option in Raspberry Pi Imager, then selecting the required Legacy image.

Raspberry Pi OS (Legacy) will remain supported while the various components continue to receive updates. For Debian Buster, support will be available until June 2024; for the Linux 5.10 kernel, December 2026. If Debian Bookworm becomes stable in this time, Raspberry Pi OS (Legacy) will switch to Bullseye.

Raspberry Pi