Using OTG mode on Raspberry Pi SBCs

Colophon

Release	1
Build date	01/10/2025
Build version	99a8b0292e31

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	1 Oct 2025	Initial release

Scope of document

This document applies to the following Raspberry Pi products:

Single Board Computers / SBCs

Pi Zero			Pi Zero 2		Pi 1		Pi 2		Pi 3	Pi 4	Pi 5
-	W	H	W	H	A	B	A	B	B	-	-
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Compute Modules

CM1	CM3	CM4	CM5
✓	✓	✓	✓

Introduction

USB On-The-Go (OTG) is a specification that allows a device to function either as a **USB host** (like a PC) or as a **USB device/peripheral** (like a keyboard, an Ethernet adapter, or a mass storage device). The 'USB On-The-Go' Wikipedia page provides a lot of detail on the OTG specification: https://en.wikipedia.org/wiki/USB_On-The-Go.

Usually, a USB connection involves a fixed host (e.g. a computer) and a peripheral (e.g. a mouse). USB OTG allows a device to switch between the two. For example, a Raspberry Pi could act as a host when reading files from a flash drive, or function as a flash drive itself when connected to a computer.

The Raspberry Pi family includes several boards that can function in **OTG/peripheral mode**, but support differs depending on the model and the system on chip (SoC). When acting in this peripheral mode, the device is often referred to as a 'gadget'.

This whitepaper walks through the Raspberry Pi SBC lineup, explains their OTG capabilities, and provides configuration/code examples. It covers two distinct OTG mechanisms: the legacy method, which is still very popular and is described first, followed by the currently recommended scheme, `ConfigFS` .

Legacy OTG

Raspberry Pi Zero / Zero W / Zero 2 W

These boards are the most **OTG-friendly** in the Raspberry Pi family.

They expose the SoC's USB controller directly on the **USB data port** (the one labelled `USB` , not `PWR IN`), and the onboard software can be configured to make your Raspberry Pi act as an OTG device.

Enabling OTG mode

Tip

Because you are using the only USB port on Raspberry Pi Zero for OTG purposes, you will not be able to plug in a keyboard or a mouse. You can instead use a Wi-Fi connection and SSH to communicate with Raspberry Pi Zero. Enable Wi-Fi, then log in to your local network, and from a different device (e.g. another Raspberry Pi, a Linux machine, or a Windows PC), use SSH to log in to Raspberry Pi Zero. Alternatively, you could use `Pi-Connect` instead of SSH. (https://www.raspberrypi.com/software/connect/

First, we need to tell the USB system to use the DWC2 driver, which will enable OTG mode. Edit `/boot/firmware/config.txt` and add the following:

Code

```
dtoverlay=dwc2
```

Now we need to configure the software to connect the requested OTG driver to the USB system; in this case, we want our Raspberry Pi Zero to appear as an Ethernet device. Edit `/boot/firmware/cmdline.txt` and add `modules-load=dwc2,g_ether` after `rootwait` , like so:

Code

```
console=serial0,115200 console=tty1 root=PARTUUID=xxxxxxxx-02 rootfstype=ext4 fsck.repair=yes rootwait modules-load=dwc2,g_ether
```

By putting this on the kernel command line, the module will be loaded during kernel boot. See the next section on how to load the module from the terminal.

Reboot, and your Raspberry Pi Zero will appear as a USB Ethernet gadget when plugged into a PC.

Other gadget modules

Instead of `g_ether` , you can try:

g_serial
: Appears as a USB serial device

g_mass_storage
: Exposes an image file as a flash drive

g_composite
: Emulates a composite device

This is not an exhaustive list of possible OTG gadget modes.

Tip

A USB composite device is a single physical device that functions as multiple independent devices to a computer, appearing as several separate interfaces or device classes. It combines different functionalities, such as a keyboard and a mouse, or a storage drive and a webcam, into a single USB device and connector. When connected, the operating system recognises and uses separate drivers for each of the device's distinct functions, allowing them to operate independently.

To create a USB serial gadget, we can load the appropriate module from the command line:

```
Code
sudo modprobe g_serial
```

When connected to a Windows PC, the Raspberry Pi will appear as a COM port in the Device Manager; when connected to a Linux device (e.g. a Raspberry Pi SBC), it will appear as a serial device like `/dev/ttyACM0` .

Raspberry Pi 4 and 5 (OTG on the USB-C power port)

Raspberry Pi 4's USB-C power/OTG port supports peripheral mode when it is not being used to power the board.

Raspberry Pi 5 introduces a PCIe-attached USB controller, which does not support OTG. However, as with Raspberry Pi 4, the native OTG peripheral function on the SoC is exposed through the power connector.

Steps

Power your Raspberry Pi through the GPIO header (5V and GND), leaving the USB-C free.

Connect the USB-C port to your host computer.

Enable OTG in `/boot/firmware/config.txt` :

```
Code
dtoverlay=dwc2,dr_mode=peripheral
```

Note

You need the `dr_mode=peripheral` option on the overlay to force the controller into OTG peripheral (rather than host) mode, as the `OTG_ID` line that would normally do the selection is not present on Raspberry Pi 4 or 5.)

Load a gadget module (Ethernet):

```
Code
sudo modprobe g_ether
```

Your Raspberry Pi will now enumerate as a USB device to the host.

Tip

Not all host systems handle Raspberry Pi 4's OTG mode reliably. Ethernet and serial work best.

Raspberry Pi Compute Module series

Raspberry Pi Compute Module 1, 3, 3+ and 4 expose the SoC's USB OTG controller directly to the carrier board, making them highly flexible.

CM1/CM3/CM3+ The USB OTG interface is available on dedicated pins; carrier boards often expose this via a micro-USB port.

CM4 Offers an OTG-capable USB 2.0 interface (USB_OTG). This is routed to the Compute Module 4 IO Board's micro-USB connector.

CM4 OTG example (Ethernet gadget)

Plug a micro-USB cable into the USB port on the IO Board.

In `/boot/firmware/config.txt` , add:

```
Code
dtoverlay=dwc2,dr_mode=peripheral
```

In /boot/cmdline.txt, add:

Code

```
modules-load=dwc2,g_ether
```

Reboot. Compute Module 4 will now appear as a USB Ethernet adapter.

Raspberry Pi A, B, B+, 2B, 3B, 3B+

The USB ports on these models are connected through a hub chip (LAN9512/LAN9514 or VIA Labs), which strips away OTG capabilities. They can only operate as USB hosts, so no OTG support is available.

Using the various device types

This section describes how to set up the most common gadget modes.

Mass storage devices

In order to use a Raspberry Pi as a mass storage device (like a USB stick), you will need to create a backing file to hold the stored data:

Code

```
# Example: Make a 256 MB file to act as "USB stick"
sudo dd if=/dev/zero of=drive.bin bs=1M count=256
# Create a VFAT file system on the backing store
sudo mkfs.vfat drive.bin
```

Edit `/etc/modprobe.d/g_mass_storage.conf` to tell the system to use the backing store:

Code

```
options g_mass_storage file=/drive.bin stall=0 removable=1
```

You can examine the contents of the backing store by mounting it on your Raspberry Pi. Here we mount it in a folder called `mountpoint` :

Code

```
sudo mkdir mountpoint
sudo mount -o loop drive.bin mountpoint
```

You'll need to adjust the paths as appropriate.

Ethernet devices

When the `g_ether` device is plugged into a Linux host, it will usually appear as a network interface named `usb0` (when using `ifconfig`).

You can (usually) connect to the device using SSH, as follows:

Code

```
ssh pi@raspberrypi.local
```

Serial devices

When a Raspberry Pi is set up as a `g_serial` device, a new serial device will appear (when using Raspberry Pi OS Bookworm with a 6.12.34 kernel, this was `/dev/ttyGS0`). When that Raspberry Pi device is then plugged into a (for example, Linux) host, the device will be recognised as a CDC ACM–compliant device and will appear as another serial port. For example, on a Raspberry Pi 500 running Bookworm, it appears as `/dev/ttyACM0` .

Under Linux, you can test the serial link by using `screen` on each device. If using Windows on the host, something like Putty should work well.

On your Raspberry Pi:

```
Code
screen /dev/ttyGS0
```

On a Linux host:

```
Code
screen /dev/ttyACM0
```

Then type something into each window — the output should appear on the other screen instead.

Note

If `screen` is not installed, use `sudo apt install screen` in a terminal window.

It's easy to see how this functionality could be used to provide a serial interface to a Raspberry Pi device that monitors a number of sensors (e.g. via I2C or SPI) and passes the collated information back, via the serial port, to the host computer.

ConfigFS/usb_gadget: a brave new world

Although they are by far the most common way to set up OTG on Raspberry Pi devices, the mechanisms described above have actually been superseded by something called `usb_gadget` , which is part of **ConfigFS**.

ConfigFS is a Linux kernel interface (a virtual file system mounted at `/sys/kernel/config`) used to configure kernel objects — including USB gadget drivers — in a modular way. Using `ConfigFS` / `usb_gadget` is more flexible than the old `g_mass_storage` / `g_ether` method, because you can compose multiple USB functions (e.g. Ethernet + serial + mass storage) at once.

However, this extra functionality does come with a higher setup cost.

The basic idea is that a set of virtual folders and files is created under the `/sys/kernel/config` folder, which defines the gadget required.

Some kernel documentation on usb_gadgets is available here: https://docs.kernel.org/driver-api/usb/gadget.html and https://www.kernel.org/doc/Documentation/ABI/testing/configfs-usb-gadget.

Setup

Setting up the DWC USB peripheral is the same as in legacy mode. Edit `config.txt` as sudo and add:

Code

```
dtoverlay=dwc
```

Now add the required module for loading on startup. Edit `/etc/modules` as sudo and add the following to the end of the file:

Code

```
dwc2
```

On Raspberry Pi OS, `ConfigFS` is already mounted at `/sys/kernel/config` , but if your OS does not have this by default, do the following:

Code

```
sudo mount -t configfs none /sys/kernel/config
```

You'll need to edit `fstab` if you want this mount to happen automatically at startup.

Now we need to configure our USB gadget by loading the `libcomposite` module — the kernel library that `ConfigFS` uses to create USB gadgets — as follows:

Code

```
sudo modprobe libcomposite
```

We can check that it has loaded properly by looking at the contents of `/sys/kernel/config` , which should now contain a folder called `usb_gadget` .

The creation of the actual USB gadget is next, which involves making a folder with the name of the gadget and then creating a set of entries within that folder to define the gadget's properties. This bash script excerpt does most of the required setup:

Code

```
cd /sys/kernel/config/usb_gadget
sudo mkdir MyGadget
cd MyGadget
echo 0x1d6b > idVendor                          # Linux Foundation's USB VID
echo 0x0104 > idProduct                         # Multifunction Composite Gadget
echo 0x0100 > bcdDevice                         # v1.0.0
echo 0x0200 > bcdUSB                            # USB2
mkdir strings/0x409                             # 0x409 = English
# Insert your own manufacturer and gadget name here
echo "RaspberryPi" > strings/0x409/manufacturer
echo "MyPiGadget" > strings/0x409/product
# Make the serial number the same as the Raspberry Pi serial number
```

```
SERIAL=`cat /proc/cpuinfo | awk '/Serial/ {print $3}' `
echo $SERIAL > strings/0x409/serialnumber
mkdir configs/c.1
echo 120 > configs/c.1/MaxPower
mkdir configs/c.1/strings/0x409
echo "Config 1" > configs/c.1/strings/0x409/configuration
```

Now that the basic device data has been set up, we need to tell the device exactly what it is. Creating each device is as simple as creating a folder in the `ConfigFS` gadget's functions folder and then linking that folder to the configuration entry within the same gadget.

Serial (CDC ACM):

Code

```
mkdir functions/acm.usb0
ln -s functions/acm.usb0 configs/c.1/
```

Ethernet (RNDIS and ECM):

Code

```
mkdir functions/ecm.usb0
ln -s functions/ecm.usb0 configs/c.1/
```

Mass storage:

As with the legacy setup, we need a backing store for our mass storage gadget:

Code

```
cd ~
# Example: Make a 256 MB file to act as "USB stick"
sudo dd if=/dev/zero of=drive.bin bs=1M count=256
# Create a VFAT file system on the backing store
sudo mkfs.vfat drive.bin
```

And to use that:

Code

```
mkdir functions/mass_storage.usb0
echo "/home/pi/drive.bin" > functions/mass_storage.usb0/lun.0/file
ln -s functions/mass_storage.0 configs/c.1/
```

Tip

You can have more than one backing store — just assign each backing store to lun.0, lun.1, lun.2, and so on.

Finally, we need to link the gadget to the USB device controller (UDC).

Code

```
UDC=`ls /sys/class/udc`
echo $UDC > UDC
```

Tip

`/sys/class/udc` is a directory within the `sysfs` file system that represents the available USB device controllers (UDCs). It allows the kernel's USB gadget subsystem to identify and interact with hardware UDCs on a device, enabling the system to function as a USB peripheral. You can list its contents using `ls /sys/class/udc/` to find the name of the UDC, such as `3f980000.usb` , and then write that name to a gadget's configuration to bind the gadget to the UDC.

Once the setup is complete, the folder structure and contents should resemble the example below, which sets up both a serial gadget and an Ethernet gadget on the same device:

Code

```
/sys/kernel/config/usb_gadget/MyGadget/
├── configs/
│   └── c.1/
│       ├── acm.usb0 -> ../../functions/acm.usb0
│       └── ecm.usb0 -> ../../functions/ecm.usb0
├── functions/
│   ├── acm.usb0/
│   └── ecm.usb0/
├── strings/
│   └── 0x409/
│       ├── manufacturer: "RaspberryPi"
│       ├── product: "MyPiGadget"
│       └── serialnumber: "12345678"
├── idVendor: 0x1d6b
├── idProduct: 0x0104
└── UDC: "3f980000.usb"
```

Now reboot, then connect the Raspberry Pi device to a host device (e.g. another Raspberry Pi, a Windows PC, or a Linux PC). The host should have a USB Ethernet device and a serial device attached.

Making it all work

All the commands described above need to be run every time the Raspberry Pi device starts up. Since Raspberry Pi OS uses `systemd` , that is the appropriate way to run the startup script that does all the setting up. Here is an example script that collates all the instructions from above:

Code

```
# Load the gadget module
sudo modprobe libcomposite
cd /sys/kernel/config/usb_gadget

# Create our new gadget
sudo mkdir MyGadget
cd MyGadget

# Add some boilerplate data that the gadget needs
echo 0x1d6b > idVendor                # Linux Foundation's USB VID. Replace with your own VID if required
echo 0x0104 > idProduct               # Multifunction Composite Gadget
echo 0x0100 > bcdDevice               # v1.0.0
echo 0x0200 > bcdUSB                  # USB2
mkdir strings/0x409                   # 0x409 = English

# Insert your own manufacturer and gadget name here
echo "RaspberryPi" > strings/0x409/manufacturer
echo "MyPiGadget" > strings/0x409/product

# Make the serial number the same as the Raspberry Pi serial number; you can use whatever is required here
SERIAL=`cat /proc/cpuinfo | awk '/Serial/ {print $3}' `
echo $SERIAL > strings/0x409/serialnumber

# Create a configuration area and populate
mkdir configs/c.1
echo 120 > configs/c.1/MaxPower

mkdir configs/c.1/strings/0x409
echo "Config 1" > configs/c.1/strings/0x409/configuration

# Create our specific gadget type – this example is a mass storage device, but you can make various different
types
mkdir functions/mass_storage.usb0

# Set up our backing store
echo "/home/pi/drive.bin" > functions/mass_storage.usb0/lun.0/file
+
# Link our function to our configuration
ln -s functions/mass_storage.0 configs/c.1/
```

This script will need to be marked as executable:

Code

```
chmod +x <script name>
```

We now need to tell `systemd` to run our script on startup.

Create a file in `/lib/systemd/system` — the name you choose is up to you (as long as the suffix is `.service`), but for this example, we will use `mass-storage-device.service` . Enter the following into the file (note that there are many different options for these service files; we've just used the ones we need):

Code

```
[Unit]
Description=Mass Storage Device
After=systemd-user-sessions.service

[Service]
ExecStart=/home/pi/gadget.sh

[Install]
WantedBy=multi-user.target
```

You will need to change the `ExecStart` line to point to wherever you have saved the setup script.

You then need to tell `systemd` to run the service on startup:

Code

```
sudo systemctl enable mass-storage-device.service
```

Now when you plug your Raspberry Pi into a host, it should appear as a mass storage device.

You can disable the `systemd` service as follows:

Code

```
sudo systemctl disable mass-storage-device.service
```

Attaching a login console to a serial port

If you have set up your Raspberry Pi as a serial gadget, you might want to use that serial gadget to log in to the device, rather than just using it for point-to-point serial communication. On the latest version of Raspberry Pi OS running `systemd` this is easy. You need to tell the system to create a `getty` on the serial port, and then tell `systemd` to start it up. The following sets up the `getty` on `ttyGS0` (The `tty` created when using `ConfigFS` to set up a serial device); you may need to adjust this to match whichever `tty` the serial device is assigned to.

Code

```
systemctl enable serial-getty@ttyGS0.service
systemctl start serial-getty@ttyGS0.service
```

This will start up the `getty` on the serial port and ensure it starts up automatically on each reboot.

Tip

What is a `getty` ? In Linux, a `getty` is a program that manages terminals (both physical serial ports and virtual consoles) to allow multiple users to log in to a system, handling tasks like initialising the terminal, displaying a login prompt, and invoking the login program to authenticate the user.

This feature can be particularly useful on something like a Raspberry Pi Zero or Raspberry Pi Zero 2 W. With just one USB connection providing both power and serial communication, you can plug in the device and log in to it via a terminal.

Conclusion

For true USB gadget projects (e.g. Ethernet, serial, mass storage), the Raspberry Pi Zero family and Raspberry Pi Compute Modules are the best choice.

Raspberry Pi 4 and Raspberry Pi 5 do offer OTG support, but their power requirements may be an issue.

Raspberry Pi A, B, 2B, 3B and 3B+ boards do not support OTG.

If your project depends heavily on OTG, the best options are Raspberry Pi Zero 2 W or Raspberry Pi Compute Module 4 with the Compute Module 4 IO Board.

There are two options on the software side: the legacy system is still commonly used and is easy to set up; the `ConfigFS` system requires more work to set up but does provide better functionality.

Quick reference table

Model	OTG Support	Notes
Raspberry Pi Zero / Zero W / Zero 2 W	Yes	Fully supported on USB data port
Raspberry Pi 4	Yes [1]	USB-C port in device mode
Raspberry Pi 5	Yes [1]	USB-C port in device mode
Raspberry Pi A/B/2B/3B/3B+	No	Only host mode
Raspberry Pi Compute Module 1–3	Yes	Exposed on OTG pins
Raspberry Pi Compute Module 4	Yes	micro-USB on CM4 IO board

[1] Raspberry Pi 4 and 5 will usually draw power from the host via the USB cable, so there may be limitations on available current due to the higher power requirements of these devices.

Contact Details for more information

Please contact applications@raspberrypi.com if you have any queries about this whitepaper.

Web: www.raspberrypi.com

Raspberry Pi

Raspberry Pi Ltd