USB Mass Storage Gadget

A Beginner's Guide

Table of Contents

1 Colophon

2 Introduction

This is a guide to using the USB mass storage gadget functionality on Raspberry Pi models that support it.

It is assumed that the reader has a basic familiarity with the Linux command line, at least one text editor, and with partitioning and formatting discs. Desktop users will need to open a terminal to execute many of the commands in this guide.

2.1 What's Not Covered

- Pi 400[1]
- Compute modules.

2.2 Requirements:

- Raspberry Pi (any model) and the normal accessories to act as the USB host.
- A Raspberry Pi zero, zeroW, zeroWH, A, A+, 3A+, or 4B and the normal accessories.
- A user account on both Pi with permission to use `sudo` or the ability to login as root.
- An appropriate USB cable:
  - zero, zeroW, zeroWH: USB A male to micro B male charge and sync.
  - A, A+, 3A+: USB A male to A male.
  - 4B: USB C male to USB A male.
- Optional but highly recommended for troubleshooting:
  - zero and zeroW: GPIO headers fitted.
  - Three female to female jumper (Dupont) wires.
  - A second channel into the Pi running in gadget mode.
    - zeroW(H), 3A+: the onboard WiFi or serial port.
    - 4B: the onboard WiFi, onboard Ethernet, or serial port.
    - Others: serial port.

1 The configuration used with a 4B may work but has not been tested. The Pi 400 cannot be powered via its GPIO header.

2.3 Conventions

```
Text like this indicates input to or output from the command line.
```

`Text like this` also refers to full or partial commands but is not generally intended to be entered into the command line as is.

“SD card” refers equally to full size and micro SD cards.

“zero” refers to the Pi running the mass storage gadget. Where other models require different configurations this will be noted.

“backing store”, “shared storage” both refer to the storage exposed to the USB host from the zero.

3 The Mass Storage Gadget

3.1 What it Does Do

- Presents a drive, partition, or image file to the USB host.

3.2 What it Does Not Do

- Provide drivers to either OS.
- Present a directory on the zero to the USB host.
- Present a network share mounted on the zero to the USB host.
- Provide safe writeable storage to either the USB host or the zero while the other has any access to the backing store.[2]
- Present a USB drive (or its contents to the USB host).[3]
- Allow the USB host to access file systems (or disc formats) it does not understand.

3.3 Restrictions

The mass storage gadget cannot be used with:

- Any PI model that has more than one onboard USB port (4B excepted).
- A USB hub downstream of the zero[4].
- Any single USB port Pi model that boots from a USB device[5].

The 4B only supports running the mass storage gadget on its USB C port however there are potential issues with this:

- If your USB host cannot provide sufficient current for the 4B it will need to have its own PSU. This can be achieved by either powering it via the GPIO header or a suitable Y cable.
- When using a separate PSU for the 4B and the USB host there is a risk of back powering the host. Use a modified USB cable or adapter with the +ve line cut to avoid this.
- While the connector is USB C, the 4B only provides a USB 2 connection and does not support USB PD.
- “E marked” cables are known to cause problems with early revision 4Bs.

2 If you must have USB connected storage that has safe simultaneous write access from both sides of the USB link, use the ethernet gadget and run a samba server on the zero.
3 The 4B can do this but there is little point in it.
4 Including any HATs or HAT like devices that connect via test points to provide additional USB ports.
5 Including via USB network adapters.

3.4 How It Works And Why Write Access Is Bad

This is going to get technical.

The mass storage gadget provides a low level block device to the USB host. The host accesses the backing store by requesting reads from/writes to it of N bytes/blocks starting at address A. The OS on the USB host then processes this data to pass the requested file[6] to the application that needs it.

The mass storage gadget has no knowledge of which files and directories are being accessed. It also has no knowledge (and no way to find out) when any given file access has completed.

Writing is safe if only the writing side has the backing store mounted. It is not safe if both sides have it mounted and are allowing writing nor is it safe if one side has it mounted read/write and the other read only.

Modern OS[7] use both read and write caching[8].

Caches on on the USB host and the zero are not, and cannot be, kept in sync.

While the mass storage gadget knows which blocks/bytes have been changed by the USB host it does not, and cannot, know which files or directories those are part of. It also has no knowledge of whether all the data to be written has been sent or if there is more to come.

The mass storage gadget is unaware of any changes made by the zero.

The USB host is unaware of any changes made by the zero, the OS on the zero is unaware of any changes made by the USB host.

Because of the above, writing from one or both sides while the other has it mounted will cause problems including:

- Files deleted from one side still being present on the other.
- Files added from one side not being shown on the other.
- Files moved or renamed from one side still showing the old name or location on the other.
- File contents being overwritten.
- General file or file system corruption.

3.5 A Warning

The zero is still a full linux computer. Removing power (e.g. by disconnecting it from the USB host) without first performing a clean shutdown will lead to lost or corrupted data, both within the backing store and in the zero's OS storage.

6 Or directory listing, partition table, boot sector, etc.
7 Embedded systems (TV, data loggers, etc.) likely do not but this does not make it any safer.
8 RAM buffers to improve performance. Some writes may never make it to disc.

4 The Backing Store

The backing store for the mass storage gadget can be one of the following:

- An entire device, e.g. `/dev/sda`
- A single partition, e.g. `/dev/sda1`
- A file, which will be treated as a block device, e.g. `/srv/backing_store`

It cannot be:

- A directory.
- A network share mounted from another computer.[9]

4.1 Using An Entire Device

The USB host has access to everything on the device including its boot sector and partition table(s). If exported to the USB host as read/write, the host can trivially destroy the entire contents of the device.

Using an entire device with a zero is not recommended as it gives the USB host access to the boot and root partitions, something that is not advisable in most circumstances.

9 A file stored on a network share can be used.

4.2 Using A Partition

Using a partition restricts the USB host's access to just the specified partition. It has no access to other parts of the device including the device's partition table and boot sector.

Because there is no access to the device's partition table the USB host may see it as a bare, uninitialised drive. Formatting the partition via the OS on the zero will not fix this, the data exposed to the USB host will still lack a partition table.

Initialising and formatting the partition from the USB host writes the necessary data to the partition (partition table etc.) but that results in a drive with two partition tables – one in the normal place and one at the start of one of its partitions. Some OS do not cope well with this when reading the drive directly rather than through the mass storage gadget.

4.2.1 Creating A Partition Based Backing Store

This is best done while the OS on the SD card is not running. A second Linux computer (or a second SD card with Raspberry Pi OS installed) and a USB card reader will be required.

Detailed instructions depend on the software tools being used. The broad steps are as follows:

1. Make sure you are working on the correct device.
2. Reduce the size of the second partition to create the required amount of free space after it ends.
3. Create a new primary partition in the free space. Leave it unformatted.
4. Write the changes to disc.

4.3 Using A File

Using a file restricts access by the USB host to just that file and its contents. A file can be of an arbitrary size[10], in an arbitrary location, and have an arbitrary name..

Disc image files (.img, .iso, etc.) can be passed to the mass storage gadget as is but the USB host must understand the partition information and file systems within in order for them to be of use.

4.3.1 Creating A File Based Backing Store

There are three[11] main methods of creating a file based backing store:

1. `fallocate`
2. `dd`
3. `mkfs` (or `mke2fs`, `mkdosfs`, etc.)

`fallocate` is fast but not supported by all file systems.

`dd` is slow and supported by all file systems. The file contents depend on what was used as the if in `dd`.

Both `fallocate` and `dd` produce unformatted files i.e. they do not contain an useable file system.

With some file system types[12], `mkfs` can create the file and the file system it contains in one operation. mkfs does not create boot sectors and partition tables. Many USB hosts will see the backing store as uninitialised and unformatted.[13]

To create a 1GB file based backing store in the current directory:

- Using `fallocate`:

  ```
  fallocate -l 1g fallocate.img
  ```

- Using `dd`:

  ```
  dd if=/dev/zero of=dd.img bs=1M count=1K
  ```

  `bs` is the block/buffer size, `count` is the number of blocks desired. See `man dd`.

- Using `mkfs`:

  ```
  mkfs -t vfat -C mkfs.img 1048576
  ```

  Size is specified in blocks. Block size can vary between file systems. Some allow it to be specified.

10 Subject to the size of the file system or partition containing it and any limitations of that filesystem.
11 Excluding taking an image of an existing drive/partition/file system.
12 See man pages for mkfs, mkdosfs, mke2fs, etc.
13 Most linux USB hosts will be able to mount it using the device node for the drive e.g. /dev/sda.

4.4 Partitioning And Formatting The Backing Store

If you created the backing store with `mkfs` this step can be skipped unless you need, or want, it to contain a partition table.

The simplest method and the one least likely to cause complications with USB hosts is to pass the raw backing store file to the mass storage gadget then initialise, partition, and format from the USB host.

To initialise, partition, and format the backing store on the zero:

1. Attach the backing store to a loop device[14]:

   ```
   sudo losetup --show -fP /path/to/backing-store
   ```

   Make a note of the path returned, e.g. /dev/loop0

2. Initialise, partition and format the backing store by using the normal linux tools (fdisk, gparted, etc) on the loop device created in the previous step. Multiple partitions can be used but not all OS and USB hosts support this.

   For cross platform use currently the best choice of file system is FAT32.

3. Detach the backing store from the loop device:

   ```
   sudo losetup -d /path/to/backing-store
   ```

14 For more information on `losetup`, see `man losetup` or `losetup –help`.

4.5 Adding[15] Content To The Backing Store

4.5.1 From The Zero

1. Disconnect the zero from the USB host.
2. Attach the backing store to a loop device as above:

   ```
   sudo losetup --show -fP /path/to/backing-store
   ```

   Make a note of the path returned, e.g. /dev/loop0
3. Mount the partition(s) into your file system:
4. sudo mount /dev/loop0p1 /mnt
5. Using your normal file management tools copy files to the mounted backing store.
6. Unmount the backing store e.g.:

   ```
   sudo umount /mnt
   ```
7. Detach the backing store from the loop device:

   ```
   sudo losetup -d /path/to/backing-store
   ```

4.5.2 From The USB Host

Once the gadget has been fully configured use as you would any other USB mass storage.

15 And removing.

5 Configure The Zero

This can be done either by logging in to the zero or by mounting its SD card in a reader attached to the USB host. All paths below assume logging in to the zero, adjust as needed if using the latter method.

5.1 All Methods

1. Back up your config.txt:

```
sudo cp /boot/config.txt /boot/config.bak
```

2. Using your preferred text editor add the following to the end of `/boot/config.txt`

```
[all]

dtoverlay=dwc2,dr_mode=peripheral
```

On next boot your OS will use the dwc2 driver in the correct mode to support operation as a USB gadget.

`,dr_mode=peripheral` is optional on zero, zeroW(H), and the 4B[16]. It is mandatory on A, A+, and 3A+ as theses are hardwired to host mode and the ID pin is not present on their USB A connectors.

16 The 4B has no ID pin but defaults to device mode.

5.2 Using The g_mass_storage Module

5.2.1 Why

- It’s easy.
- It takes care of the majority of the set up and configuration for you.

5.2.2 Why Not

- It’s old.
- It’s deprecated and will be going away at some point.
- It only provides the mass storage gadget. If you want other gadgets at the same time this isn’t possible.
- It’s not possible to change the backing store without unloading and reloading the module.

5.2.3 How

1. Apply the steps in section 5.1
2. Open root’s crontab:

   ```
   sudo crontab -e
   ```

3. Add the following at the end of the file:

   ```
   @reboot /sbin/modprobe g_mass_storage file=/path/to/backing-store
   ```

   Change /path/to/backing-store as required.

   If using a partition it is possible to use LABEL, PARTUUID, etc by using one of the symlinks found in `/dev/disk/by-*/` rather than `/dev/mmcblk0p?`

4. Save and close.
5. Reboot

This will give you a mass storage gadget with the default options: rw=y, removable=n, cdrom=n.

5.2.4 Module Parameters

For full documentation see https://www.kernel.org/doc/Documentation/usb/mass-storage.txt

- `cdrom=b[,b…]`

  Specify whether a backing store is presented to the USB host by emulating a CD-ROM drive.

  `b` can be “n”, “N”, or “0” for no; “y”, “Y”, or “1” for yes. The default is no.

- `file=filename[,filename…]`

  Path to the file or block device used for the backing store. Multiple values may be specified up to the default maximum of eight[17]. If removable is not set to on, a file must be provided.

  Each file will be presented to the USB host as a separate mass storage device.

- `removable=b[,b…]`

  Specify whether a backing store is presented to the USB host as a device with removable media.

  `b` can be “n”, “N”, or “0” for no; “y”, “Y”, or “1” for yes. The default is no.

- `ro=b[,b…]`

  Specify whether a backing store is presented to the USB host as a read only device.

  `b` can be “n”, “N”, or “0” for no; “y”, “Y”, or “1” for yes.

  The default for emulated CD-ROM devices is yes, for all others it is no. If the backing store cannot be opened for writing the ro parameter will fall back to being on.

Other parameters can be left at their default values and not specifically included.

17 The ninth and subsequent files will be silently ignored.

5.3 Using The libcomposite Module And configfs

5.3.1 Why

- It gives finer control over the gadget.
- It allows more than one gadget on the same zero.
- When using multiple gadgets they can be of different types[18].

5.3.2 Why Not

- It's more complicated.
- Everything must be setup manually.
- Some USB hosts may initially report an "unknown USB device" during the zero's boot process only to correct this once gadget configuration is complete.

5.3.3 How

This is a step by step guide as would be typed at a terminal. In practise you'll probably want to put this in a script[19].

1. Apply the steps in section 5.1
2. Become root:

```
sudo -i
```

3. Load libcomposite:

```
modprobe libcomposite
```

4. Create a directory for the gadget configuration:

```
mkdir -p /sys/kernel/config/usb_gadget/mygadget
```

5. Make that the current directory:

```
cd /sys/kernel/config/usb_gadget/mygadget
```

6. Set the IDs for the gadget:

```
echo 0x1d6b > idVendor
echo 0x0104 > idProduct
echo 0x0100 > bcdDevice
echo 0x0200 > bcdUSB
```

18 Using multiple gadgets is outside the scope of this guide.
19 Except the first two and the last one.

7. Configure text strings:

```
mkdir -p strings/0x409
echo "1234567890" > strings/0x409/serialnumber
echo "me" > strings/0x409/manufacturer
echo "My USB Device" > strings/0x409/product
```

   If you know it, use the zero's serial number. Change the other strings to taste.

8. Initial device configuration:

```
mkdir -p configs/c.1/strings/0x409
echo "Config 1: Mass Storage" > configs/c.1/strings/0x409/configuration
echo 250 > configs/c.1/MaxPower
```

9. Configure the mass storage gadget:

```
mkdir -p functions/mass_storage.usb0
echo 0 > functions/mass_storage.usb0/lun.0/cdrom
echo 0 > functions/mass_storage.usb0/lun.0/ro
echo /path/to/backing-store > functions/mass_storage.usb0/lun.0/file
ln -s functions/mass_storage.usb0 configs/c.1/
```

   Change `/path/to/backing-store` as appropriate.

   Repeat the above three echo commands adjusting lun.0 (to lun.1, lun.2 etc.) if more than one backing store is needed.

   See 5.2.4 for details of `cdrom`, `ro`, and `file`.

10. Finish up[20]:

```
ls /sys/class/udc > UDC
```

11. Exit the root shell.

20 Additional gadget configuration must be done prior to this step.

5.3.4 Bash Script Example

```
#!/bin/bash
modprobe libcomposite
mkdir -p /sys/kernel/config/usb_gadget/mygadget
cd /sys/kernel/config/usb_gadget/mygadget
echo 0x1d6b > idVendor
echo 0x0104 > idProduct
echo 0x0100 > bcdDevice
echo 0x0200 > bcdUSB
mkdir -p strings/0x409
echo "1234567890" > strings/0x409/serialnumber
echo "me" > strings/0x409/manufacturer
echo "My USB Device" > strings/0x409/product
mkdir -p configs/c.1/strings/0x409
echo "Config 1: Mass Storage" > configs/c.1/strings/0x409/configuration
echo 250 > configs/c.1/MaxPower
mkdir -p functions/mass_storage.usb0
echo 0 > functions/mass_storage.usb0/lun.0/cdrom
echo 0 > functions/mass_storage.usb0/lun.0/ro
echo /path/to/backing-store > functions/mass_storage.usb0/lun.0/file
ls /sys/class/udc > UDC
```

1. Save the script
2. Give it execute permission

   ```
   chmod a+x /path/to/script
   ```

3. Call it from root's `crontab`, `/etc/rc.local` or as a systemd service.

5.4 Verify It’s Working

1. Boot your USB host.
2. Connect the zero and allow it to boot.
3. Log in to your USB host.
4. Run

   ```
   lsblk
   ```

   The output should show an[21] additional sd device or sr[22] Device and it’s associated partitions.

21 Or one per file specified if using more than one backing store.
22 If the gadget has been configured with `cdrom=y`

6 Troubleshooting

6.1 Configure The Secondary Channel into The Zero

It may be necessary to remove or comment out[23] the `dtoverlay` line from `/boot/config.txt` and reboot in order to allow local login to the zero to perform these steps. Remember to re-enable it afterwards and reboot.

Without a secondary channel into the zero troubleshooting will be several orders of magnitude more difficult.

Once troubleshooting is complete the secondary channel can be disconnected.

6.1.1 The 4B – A Special Case

Due to the hardware of the 4B (two entirely separate USB controllers) it can function as both USB host and USB device at the same time. If you have a way to provide power to the 4B while using its USB C port for data (e.g. via the GPIO header) this ability can be used to troubleshoot on a single Pi even if the final model is not a 4B.

Use a USB A male to USB C male cable to connect the USB C and one of the USB A female ports.

6.1.2 Serial Port

Using a serial port allows the boot messages to be monitored, does not impact the network configuration, and does not require reconnection after each reboot.

It requires that the GPIO header has been fitted to the zero and that the UART pins on the USB host are free. A USB to 3.3v[24] serial adapter may be used on the USB host if the pins are not free.

No software configuration to the zero should be required. At time of writing Raspberry Pi OS defaults to having boot messages and a login available on the serial port.

1. Using a jumper wire, connect the TX (GPIO 14, physical 8) pin of the zero to the RX (GPIO 15, physical 10) pin of the USB host.
2. Using a jumper wire, connect the RX (GPIO 15, physical 10) pin of the zero to the TX (GPIO 14, physical 8) pin of the USB host.
3. Connect any ground pin of the zero to any ground pin of the USB host.
4. Login to the USB host.
5. Open `raspi-config`. You'll need to be root or use `sudo`.

23 Insert a # at the start of the line.

24 Do not use a 5v adapter, an RS232 adapter, or connect to a PC's nine pin serial port. Doing so will damage the zero.

6. Ensure a login shell is not accessible over serial. This should be under “Interface Options” then “Serial Port”.
7. Ensure the serial port is enabled.
8. Ensure that log in over the serial port is not enabled.
9. Install screen:

```
sudo apt update
sudo apt install screen
```

10. Ensure you user can access the serial ports:

```
sudo usermod -a -G pi tty
```

    Replace pi if using a different user.

11. Reboot.
12. You can now open the serial port:

```
screen /dev/serial0 115200
```

Hit `control-a` then `k` to exit.

6.1.3 Network

Zero, A, and A+ models lack onboard networking so cannot use this method.

ZeroW, zeroWH, 3A+, and 4B models can use their onboard WiFi.

The 4B can also use its onboard ethernet.

Configuration is straight forward:

1. Set up networking on the zero in the usual way.[25]
2. Enable ssh on the zero.[26]
3. Reboot the zero.

25 See https://www.raspberrypi.org/documentation/configuration/tcpip/README.md and https://www.raspberrypi.org/documentation/configuration/wireless/README.md

26 See https://www.raspberrypi.org/documentation/remote-access/ssh/README.md

6.1.4 Serial Over USB

This needs some changes to your gadget configuration. The connection will also be lost during reboots of the zero and boot messages will not be visible[27].

This is not recommended on Windows USB hosts due to driver issues.

1. Reconfigure the gadget:

   If using g_mass_storage:

   Switch to libcomposite/configfs

   If using libcomposite/configfs:

   Add the following above the line that reads "`ls /sys/class/udc > UDC`" in your configuration script:

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

2. Enable login via the gadget serial port:

   ```
   sudo systemctl enable getty@ttyGS0.service
   ```

3. Reboot.

You can now login to the zero with `screen`.[28] Use `/dev/ttyACM0` rather than `/dev/serial0`. If that fails check which serial device has been assigned by the OS.

27 Unlike when using a true serial port.
28 See section 6.1.2 for details.

6.2 The Zero Does Not Boot

6.2.1 All cases

- Check the cable including any adapters.
- Connect a monitor and view the boot messages, if any.
- Check for errors in `/boot/config.txt`
- If using libcomposite/configfs check for errors in your configuration script.
- Follow the usual boot failure trouble shooting steps. See https://www.raspberrypi.org/forums/viewtopic.php?f=28&t=58151
- To rule out hardware failure, try to boot from a different SD card containing a fresh OS installation. If it still fails to boot it's likely a hardware issue with the zero.

6.2.2 Power From USB Host Only

- Check the USB host's power supply can provide enough current. See https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md
- 4B gadget only: switch to a "Y" cable[29] (2 x USB A male, 1 x USB C) between the USB host and the 4B or add a separate PSU. The 4B requires a minimum of 600mA on a bare board. This exceeds the 500mA upper limit for a USB 2 device.

6.2.3 Zero With Its Own PSU

Check the power supply can provide enough current. See https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md

Using an unmodified USB cable between the zero and the USB host in this instance may be problematic. If voltages are not exactly the same one will try to feed power to the other.

29 If the USB host is a Raspberry Pi this will make no difference. No current model Pi does per port USB current limiting.

6.3 The USB Host Does Not Detect The Mass Storage Gadget

- Check that the zero has booted.
- Check the gadget has loaded on the zero.
- Check your modprobe command or configfs script for errors.
- Check the cable.

6.4 The USB Host Cannot Mount The Backing Store

The exact cause will vary but the error return by the mount command may help.

- If the backing store was created with mkfs and has no partition table, try recreating it with a partition table. See 4.4
- Make sure the correct device node is being used. This must be the one reported by the USB host.
- Make sure the backing store has been formatted with a file system the OS on the USB host understands.
- If the backing store has not been initialised, partitioned, and formatted do so using the tools provided by the USB host[30]. Be aware that this will destroy all data within the backing store.
- If the gadget was configured with `removable=1`, make sure a backing store was also provide via `file=`.[31]

30 Windows users may need to do this in disk management rather than in explorer.
31 Or the equivalents if using configfs

6.5 The USB Host Cannot Write To The Backing Store

In order for the USB host to be able to write to the backing store the following must all be true:

1. The zero's SD card must not have failed and gone read only.
2. The partition containing a file based backing store must not be mount read only on the zero.
3. A file based backing store must have write permission for the *root* user.
4. The mass storage gadget must not be configured with `ro=y` or `cdrom=y`.
5. The USB host must mount the gadget as read/write.

6.6 Changes Made To The Contents Of The Backing Store On One Side Are Not Visible On The Other

This expected and normal behaviour. It is also not recommended. See 3.4

Unmount and remount on the side (host or zero) that cannot see the changes. Any pending changes in the other side’s write cache will still not be visible.

To be sure that all changes have been written and that no changes will be made behind the back of the reading side, unmount the writing side before (re)mounting on the reading side. This must be done manually, neither side has any way to force the other to unmount.

7 Advanced Usage

7.1 Making A File Based Backing Store More Secure

This applies only to access from a user logged in to the zero and assumes the backing store is a file on a linux native file system[32]. It has no impact on access to the contents of the backing store once it has been mounted on the zero or the USB host.

Default file permissions allow any user to read a file, its owner to write to it, and its owner (or root) to delete it. This is likely not desirable, especially if its owner is a normal user. The following steps improve this:

1. Change the file so it is owned by root and has a group of root:

   ```
   sudo chown root:root /path/to/backing-store
   ```

2. Change permissions so that only root has access to it:

   ```
   sudo chmod og-rwx /path/to/backing-store
   ```

3. Make the file immutable[33]:

   ```
   sudo chattr +i /path/to/backing-store
   ```

32 One that fully supports linux owner, group, permissions and attributes e.g ext4.

33 This prevents accidental deletion.

7.2 Changing The Backing Store On The Fly

While possible, this not really recommended. The zero has no way to know whether the host is in a suitable state that it is safe to do so nor does it have any way to force the host to unmount the backing store.

The risks and dangers are the same as those when disconnecting any drive without first unmounting it.

7.2.1 When Using g_mass_storage

1. Unload the g_mass_storage module:

```
sudo rmmod g_mass_storage
```

2. Allow the USB host to update
3. Reload the module with different parameters:

```
sudo modprobe g_mass_storage file=/path/to/different-backing-store
```

7.2.2 When Using libcomposite and configfs

1. Ensure you have configured the gadget with removable=1
2. Remove the current backing store:

```
sudo echo "" >
   /sys/kernel/config/usb_gadget/functions/mass_storage.usb0/lun.0/file
```

3. Allow the USB host to update.
4. Set the new backing store:

```
sudo echo "/path/to/different-backing-store" >
   /sys/kernel/config/usb_gadget/functions/mass_storage.usb0/lun.0/file
```

7.3 Putting The Backing Store On A Network Server

This is only possible on Pi models that have onboard networking that does not use USB. At time of writing that was zeroW, zeroWH, 3A+, and 4B.

7.3.1 File Based Backing Store

1. Configure as for a local file based backing store.
2. Ensure the network share containing the backing store has been mounted by the zero before configuring the gadget or loading the g_mass_storage module.

NFS exports may need the `norootsquash` option.

7.3.2 Partition Or Device Based Backing Store

While possible this is significantly more complex and requires the use of a SAN[34] protocol and server. As such it is outside the scope of this guide.[35]

34 **S**torage **A**rea **N**etwork

35 But here's one I made earlier: https://www.instructables.com/NAS-Access-for-Non-Networked-Devices/

7.4 Making The Zero More Robust

7.4.1 Adding a Shutdown Button

A shutdown button on the zero will have no effect on the USB host so data written by it to the backing store may still be lost. It will, however, reduce the risk of damage to the zero's OS due to improper shutdown if used.

1. Connect a momentary, push to make button between GPIO 3 (physical pin 5) and any ground pin.
2. As root (or with sudo) open `/boot/config.txt` in your preferred text editor.
3. Add the line

   ```
   dtoverlay=gpio-shutdown
   ```

4. Save and close.
5. Reboot.

GPIO 3 cannot be used if I2C is enabled.

Refer to `/boot/overlays/README` for full details on the overlay including how to use a different GPIO.

7.4.2 Make The Root Partition Read Only

A read only root partition protects the OS on the zero. It does not provide any additional protection to the backing store.

This is trivial if using a device or partition based backing store but more complex when using a file based one if write access to it is required.

1. If the gadget is active, shut it down or at a minimum have the USB host unmount and disconnect it.
2. If write access to a file based backing store is required[36]:
    1. Follow the steps in 4.2.1 but rather than leaving the new partition unformatted format it to ext4.
    2. Create a suitable mount point for the new partition. E.G. `/srv/backing-store`
    3. Edit `/etc/fstab` and add an entry for the new partition[37]. Include the `sync` mount option.[38]
    4. Mount the new partition.
    5. Move the existing file based backing store to the new partition.
    6. Update your gadget configuration for the new location of the backing store.
3. Open the raspi-config utility: `sudo raspi-config`
4. Open “4. Performance Options”
5. Select “P3 Overlay File System Enable/disable read-only file system”
6. Enable it
7. Optionally make the boot partition read only.
8. Exit raspi-config
9. Reboot.

Remember to disable the overlay file system should you need to make changes to the root partition and to re-enable it afterwards.

36 Depending on the size of the SD card and the backing store you may need to move the backing store onto a a separate drive to carry out this procedure.

37 See `man fstab` and `man mount`.

38 Sync disables read and write caching on the partition thus reducing chance of data being lost if power is suddenly removed.