How to ... format NVMe

Background

At the MUG Christmas market, RISC OS Bits were demonstrating their NVMe driver software. It was incomplete - the module had not yet been incorporated into the rom - but looked very promising.

At the WROCC Zoom meeting in December Andy Marks explained that the NVMe driver was still being optimised but was already up to SATA speeds.

I have a Waveshare IO-Mini-B board with an NVME slot and a 512GB drive as well as CM4 with eMMc storage. I want to prepare the NVMe drive for when RISC OS can read such discs. There will be four partitions on the drive: a FAT partition for Linux firmware, a filecore partition (for RISC OS), a larger FAT partition (for file sharing between RISC OS and Linux) and an ext4 partition (for Linux).

Why?

The intention is to use the NVMe drive for both Linux and RISC OS with a switch to select which one you want to use. A 4-partition drive allows data sharing.

How do I get there?

There are several steps of preparation, which have already been described (WROCC Nov 2023 'Putting Linux on the PiRO Qube' and Archive 25:5 'Adding a switch to the 4té') that get a dual boot machine either into RISC OS with its hard disc on eMMc or into Linux (using a Store 'n' Stay Nano 32GB USB drive).

Summarising these steps, the first step is to flash the eMMc memory on the CM4 with a standard RISC OS distro with a 4 Jan 2024 daily rom (i.e. one which is eMMc-aware). Second step is to use HForm to reformat the eMMc to provide a 14GB (vice 1800MB) filecore partition and a 80MB (vice 48MB) FAT partition and to copy back the HardDisc4 files (except Loader) and the firmware files. Third step is to put a standard Bullseye Linux distro onto the USB drive and to edit the eMMc firmware files to provide a RISC OS/Linux dual boot system.

How to do this is described below and means that we have a 'dual boot' machine.

```
console=serial0,115200 console=tty1 root=PARTUUID=cf3c2bea-02 rootfstype=ext4
elevator=deadline fsck.repair=yes rootwait quiet init=/usr/lib/raspberrypi-sys-mods/
firstboot splash plymouth.ignore-serial-consoles
```

Above: *The contents of 'CMDLINE/TXT' on the standard Linux distro - the location of the root file system is specified by a partition UUID, highlighted above. For a particular disk image this magic number is known. If you specify the correct partition UUID for an ext4 partition on a drive then Linux will mount that partition as its 'root' drive '/' and then look at '/etc/fstab' for the UUID of the partition from which it will load its kernel and mount as '/boot'. The text highlighted in green will set up things like time zone and language and amend itself so that it will, on next boot, expand the 'ext4' partition to fill the drive. When it does this the UUID will change and the reference to it in '/boot/CMDLINE.TXT' and in '/etc/fstab' will be updated* ***on the same drive*** *(i.e. on the Linux boot drive not on the eMMc).*

Whereas RISC OS requires very few, if any, commands in the CMDLINE/TXT *file (*`disable_gamma` *and* `disable_mode_changes` *for example), Linux requires lots of stuff, which RISC OS can happily ignore.*

The partition UUID is a unique identifier for a partition on any drive (USB, SD, eMMc or NVMe).

Below: *The amended contents of 'CMDLINE/TXT' on the eMMc FAT partition (init to be removed).*

```
disable_gamma console=serial0,115200 console=tty1 root=PARTUUID=cf3c2bea-02
rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet init=/usr/lib/raspi-
config/init_resize.sh splash plymouth.ignore-serial-consoles
```

```
[gpio5=1]
fake_vsync_isr=1
framebuffer_swap=0
gpu_mem=64
init_emmc_clock=100000000
ramfsfile=CMOS
ramfsaddr=0x508000
kernel=RISCOS.IMG
device_tree=
hdmi_drive=2
hdmi_blanking=1
disable_overscan=1
[pi4]
enable_gic=1
[all]
[gpio5=0]
# Additional overlays and parameters are
documented /boot/overlays/README
# Enable audio (loads snd_bcm2835)
dtparam=audio=on
[pi4]
# Enable DRM VC4 V3D driver on top of the
dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2
[all]
#dtoverlay=vc4-fkms-v3d
```

The edited CONFIG/TXT file - the conditional statements are in square brackets with the RISC OS bit and Linux bit (almost empty) highlighted.

Where are we now?

Machine start up uses instructions in the eMMc firmware either to load the RISC OS ROM or (with button pressed) to mount the Linux root file system (which contains '/etc/fstab' which specifies where the Linux kernel is to found).

The changes to the firmware are summarised above. To keep things simple I shall now be making only tiny changes to the eMMc storage - a few tweaks to the file CMDLINE.TXT to identify different UUIDs for the Linux root filesystem and I shall be making these changes in RISC OS via the file Boot:Loader.CMDLINE/TXT.

We now have a 10GB Linux partition on the USB drive (with its kernel in the FAT partition on that drive) and a 15GB filecore partition on the eMMc (with its ROM in the FAT partition on that drive). For simplicity, the contents of both these FAT partitions are identical.

What next?

I want to prepare an NVMe drive but RISC OS cannot yet 'see' such drives. My solution is to use a USB 'caddy' into which I mount the NVMe drive. Suitable devices are listed below. This is the drive we are going to prepare: once we have done so then it can be mounted in its PCIe slot on the IO board (Linux will be able to see it and to mount its root filesystem on it and load its kernel from it. There will be a RISC OS filecore partition on it as well.)

In RISC OS use HForm to format a filecore partition of about 110GB (use fewer cylinders than it suggests) on the NVMe drive. Then use !SystemDisc to add a 300MB FAT partition embedded in the filecore partition.

Now boot into Linux and use GPartEd to add a 20GB FAT partition (and format it as FAT32) and a 19GB ext4 partition (and format it as 'ext4'). Also format the 300MB FAT partition as 'FAT32'.

Add a small file to each FAT partition as follows: 300MB.txt which contains:
This 300MB partition has Linux firmware
20GB.txt which contains:
This 20GB partition is for file sharing
eMMc.txt which contains:
This 80MB partition controls startup.

https://www.amazon.co.uk/dp/B07W44F9BN?psc=1&ref=ppx_yo2ov_dt_b_product_details

Aluminum M.2 NVME SSD Enclosure, USB 3.1 Gen 2 to NGFF NVME PCI-E M-Key Solid State Drive External Enclosure, With Type C-C Cable - price £15.99

https://www.amazon.co.uk/dp/B07XVR1KKR?psc=1&ref=ppx_yo2ov_dt_b_product_details

SABRENT 2242 M.2 NVMe SSD 512gb, SSD 1700MB/s Read, 42mm PCIe 3.0 X4, Internal Solid State Drive, High Performance Compatible with All PCs, NUCs, and Laptops (SB-1342-512) - price £59.99

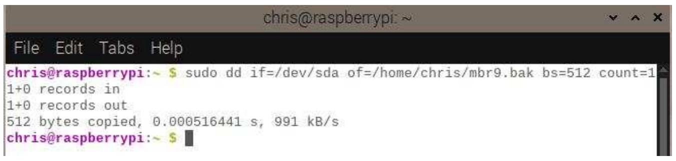
The purpose of these files is to make double sure you don't get confused as I find Linux's method of referring to different drives bizarre so the first thing to do is to make sure we know which drive is what.

Problem

The method above will first create partition 1 as FAT, embedded within the second 'ADFS' partition, numbered as partition 4, with partitions 2 and 3 empty (SystemDisc will create a Master Boot Record - MBR - describing this).

This is the standard Pi distro approach allowing the image file Boot:Loader to see the FAT partition with the firmware.

In Linux the third partition will then be created as partition 2 and the fourth partition as partition 3. This is not where we want to end up as it would confuse Linux. However there is a command which will write just the 512 bytes containing the partition table (the MBR) so that we can adjust it to list the partitions in the right order:



```
chris@raspberrypi:~ $ sudo dd if=/dev/sda of=/home/chris/mbr9.bak bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.000516441 s, 991 kB/s
chris@raspberrypi:~ $
```

The 'dd' command writes the 512 byte MBR to a file we can copy somewhere visible to RISC OS.

Using the 'dd' command we save the partition table and work out how to tinker with it to get the partitions in order:

We'll examine it using a short BASIC programme 'MBRSort' which will show us (and decode) the partition table. This programme reads the 16-byte entries in the MBR which define the start and end sector of each partition and lets us reassemble them in the correct order.

This shows us that the order on disc is p1, p4, p3 and p2 and so the programme 'MBRSort' rearranges these into this order in the MBR and the result can be written back to '/dev/sda'. After writing this, reboot immediately back into Linux.

```
 10 : REM >MBRSort
 20 : DIM mem% 3000
 30 : OSCLI("Load mbr9/bak " +STR$~(mem%))
 40 : OSCLI("Save p1 "+STR$~(mem%+&1BE)+" +10")
 50 : OSCLI("Save p4 "+STR$~(mem%+&1EE)+" +10")
 60 : OSCLI("Save p2 "+STR$~(mem%+&1CE)+" +10")
 70 : OSCLI("Save p3 "+STR$~(mem%+&1DE)+" +10")
 80 : PRINT "Partition 1"
 90 : PROCshw(&1BE)
100 : PRINT "Partition 2"
110 : PROCshw(&1CE)
120 : PRINT "Partition 3"
130 : PROCshw(&1DE)
140 : PRINT "Partition 4"
150 : PROCshw(&1EE)
160 : OSCLI("Load p1 " +STR$~(mem%+&1BE))
170 : OSCLI("Load p4 " +STR$~(mem%+&1CE))
180 : OSCLI("Load p3 " +STR$~(mem%+&1DE))
190 : OSCLI("Load p2 " +STR$~(mem%+&1EE))
200 : OSCLI("Save mbr9r/bak "+STR$~(mem%)+" +200")
210 : END
220 : DEFPROCshw(m%)
230 : PRINT "bootable?";mem%?m%
240 : PRINT "Partition type ";mem%?(m%+4)
250 : h%=mem%?(m%+1)
260 : s%=mem%?(m%+2)
270 : hpc%=136
280 : spt%=63
290 : c%=mem%?(m%+3)
300 : c%+=(s% AND &C0)<<2
310 : s%=s% AND &3F
320 : cc=1.0*c%*hpc%*h%*spt%/(s%-1)
330 : PRINT "First sector at Head ";h%;" Cyl ";c%;
340 : PRINT " Sector";s%;" LBA=";cc
350 : h%=mem%?(m%+5)
360 : s%=mem%?(m%+6)
370 : c%=mem%?(m%+7)
380 : c%+=(s% AND &C0)<<2
390 : s%=s% AND &3F
400 : cc=1.0*c%*hpc%*h%*spt%/(s%-1)
410 : PRINT "Last sector at Head ";h%;" Cyl ";c%;
420 : PRINT " Sector";s%;" LBA=";cc
430 : PRINT "LBA first sec=";mem%!(m%+8)
440 : PRINT "# sectors=";mem%!(m%+12)
450 : PRINT "LBA last sec=";mem%!(m%+12) +mem%!(m%+8)
460 : ENDPROC
```

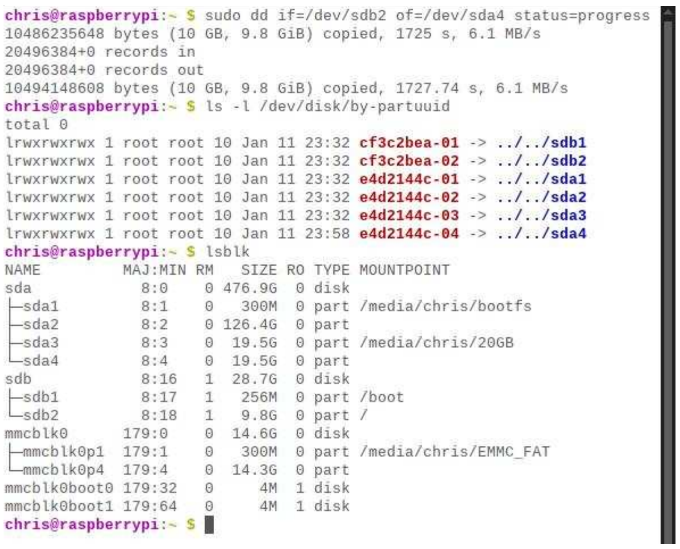
So now there are two USB drives 'sda' which is still almost blank (only an empty Boot:Loader file pointing to the 300MB FAT partition) plus a large blank FAT partition and a large blank ext4 partition and 'sdb' that has a standard 'Linux' distro on it.

Now we can copy the Linux root partition from '/dev/sdb2' to '/dev/sda4' using the 'dd' command. We can also copy the firmware files in '/dev/sdb1' to '/dev/sda1'. The UUID of the two 'Linux' partitions will be different so we need to

```
>RUN
Partition 1
        bootable?128
        Partition type 11
        First sector at Head 0 Cyl 0 Sector0 LBA=0
        Last sector at Head 0 Cyl 0 Sector0 LBA=0
        LBA first sec=176
        # sectors=614400
        LBA last sec=614576
Partition 2
        bootable?0
        Partition type 131
        First sector at Head 254 Cyl 1023 Sector2
        LBA=2.22632626E9
        Last sector at Head 254 Cyl 1023 Sector2
        LBA=2.22632626E9
        LBA first sec=306569216
        # sectors=40960000
        LBA last sec=347529216
Partition 3
        bootable?0
        Partition type 11
        First sector at Head 254 Cyl 1023 Sector2
        LBA=2.22632626E9
        Last sector at Head 254 Cyl 1023 Sector2
        LBA=2.22632626E9
        LBA first sec=265609216
        # sectors=40960000
        LBA last sec=306569216
Partition 4
        bootable?0
        Partition type 173
        First sector at Head 0 Cyl 0 Sector0 LBA=0
        Last sector at Head 0 Cyl 0 Sector0 LBA=0
        LBA first sec=614576
        # sectors=264993424
        LBA last sec=265608000
>
```

Above: *The output from 'MBRSort' - partitions 1 and 4 are actually embedded together.*

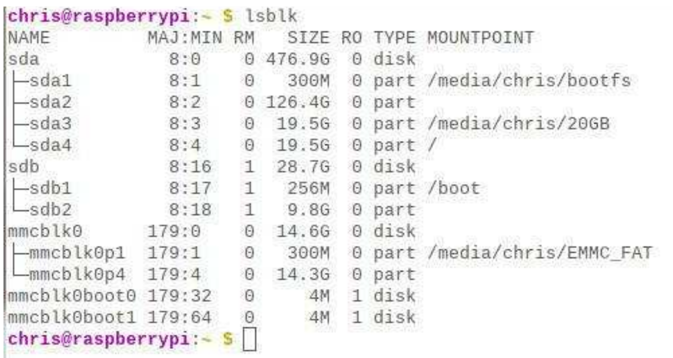
Below: *here we copy the 'Linux' partition from pen drive to NVMe drive (in its USB caddy), then list the UUIDs of each USB drive and then use 'lsblk' to show that the Linux root file system and boot drive are still both on the pen drive 'sdb'. After a reboot, sda4 will also be mounted.*



```
chris@raspberrypi:~ $ sudo dd if=/dev/sdb2 of=/dev/sda4 status=progress
10486235648 bytes (10 GB, 9.8 GiB) copied, 1725 s, 6.1 MB/s
20496384+0 records in
20496384+0 records out
10494148608 bytes (10 GB, 9.8 GiB) copied, 1727.74 s, 6.1 MB/s
chris@raspberrypi:~ $ ls -l /dev/disk/by-partuuid
total 0
lrwxrwxrwx 1 root root 10 Jan 11 23:32 cf3c2bea-01 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Jan 11 23:32 cf3c2bea-02 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Jan 11 23:32 e4d2144c-01 -> ../../sda1
lrwxrwxrwx 1 root root 10 Jan 11 23:32 e4d2144c-02 -> ../../sda2
lrwxrwxrwx 1 root root 10 Jan 11 23:32 e4d2144c-03 -> ../../sda3
lrwxrwxrwx 1 root root 10 Jan 11 23:58 e4d2144c-04 -> ../../sda4
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    0 476.9G  0 disk
├─sda1         8:1    0   300M  0 part /media/chris/bootfs
├─sda2         8:2    0 126.4G  0 part
├─sda3         8:3    0  19.5G  0 part /media/chris/20GB
└─sda4         8:4    0  19.5G  0 part
sdb            8:16   1  28.7G  0 disk
├─sdb1         8:17   1   256M  0 part /boot
└─sdb2         8:18   1   9.8G  0 part /
mmcblk0      179:0    0  14.6G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part /media/chris/EMMC_FAT
└─mmcblk0p4  179:4    0  14.3G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
chris@raspberrypi:~ $
```

note down the UUID of the ext4 partition on the NVMe drive in its USB caddy - here it is 'e4d2144c-04' - by using the 'ls' command.

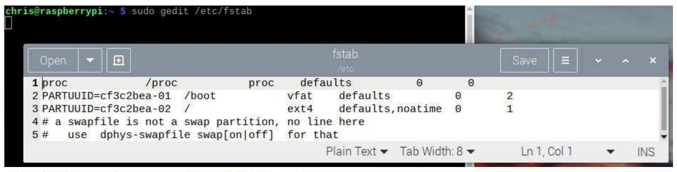
So ... if we change the UUID in the file 'CMDLINE.TXT' in the eMMc to identify this new UUID then Linux will load from the NVMe drive? Not quite. It decides where to load the kernel by reading the contents of file '/etc/fstab' on its root file system so whilst it loads its root file system from the NVMe drive it still loads its kernel from the 32GB USB drive.



```
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    0 476.9G  0 disk
├─sda1         8:1    0   300M  0 part /media/chris/bootfs
├─sda2         8:2    0 126.4G  0 part
├─sda3         8:3    0  19.5G  0 part /media/chris/20GB
└─sda4         8:4    0  19.5G  0 part /
sdb            8:16   1  28.7G  0 disk
├─sdb1         8:17   1   256M  0 part /boot
└─sdb2         8:18   1   9.8G  0 part
mmcblk0      179:0    0  14.6G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part /media/chris/EMMC_FAT
└─mmcblk0p4  179:4    0  14.3G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
chris@raspberrypi:~ $
```

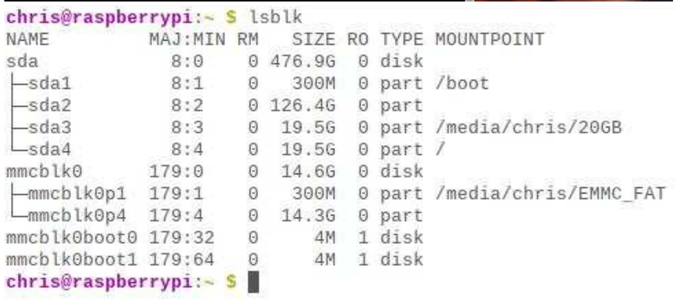
Linux is still loading the kernel from the USB drive '/dev/sdb1'

So we need to edit the file '/etc/fstab' using the command 'sudo gedit /etc/fstab' to specify the new UUID.



```
chris@raspberrypi:~ $ sudo gedit /etc/fstab
```

```
proc            /proc           proc    defaults          0       0
PARTUUID=cf3c2bea-01  /boot           vfat    defaults          0       2
PARTUUID=cf3c2bea-02  /               ext4    defaults,noatime  0       1
# a swapfile is not a swap partition, no line here
#   use  dphys-swapfile swap[on|off]  for that
```

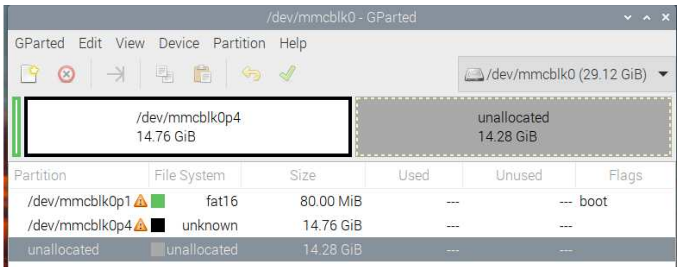


```
chris@raspberrypi:~ $ lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda            8:0    0 476.9G  0 disk
├─sda1         8:1    0   300M  0 part /boot
├─sda2         8:2    0 126.4G  0 part
├─sda3         8:3    0  19.5G  0 part /media/chris/20GB
└─sda4         8:4    0  19.5G  0 part /
mmcblk0      179:0    0  14.6G  0 disk
├─mmcblk0p1  179:1    0   300M  0 part /media/chris/EMMC_FAT
└─mmcblk0p4  179:4    0  14.3G  0 part
mmcblk0boot0 179:32   0     4M  1 disk
mmcblk0boot1 179:64   0     4M  1 disk
chris@raspberrypi:~ $
```

Now it is doing what we want and the 32GB USB drive is no longer needed (and has been unplugged)

Now if we remove the 32GB USB pen drive and reboot into Linux it should load the kernel from the NVMe drive.

Let us recap: we started with a Linux distro on a 32GB USB pen drive and a normal RISC OS distro on the eMMc storage:


/dev/mmcblk0 - GParted

GParted Edit View Device Partition Help

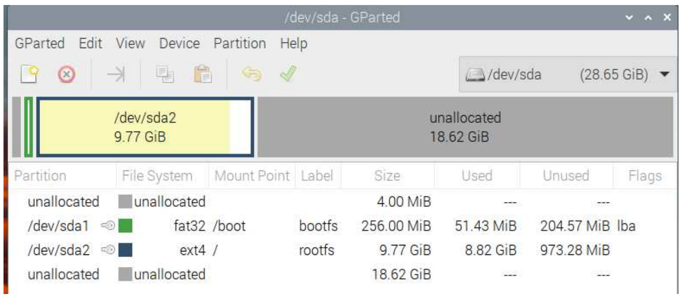
/dev/mmcblk0 (29.12 GiB)

/dev/mmcblk0p4 14.76 GiB | unallocated 14.28 GiB

Partition	File System	Size	Used	Unused	Flags
/dev/mmcblk0p1 ⚠ 🟩	fat16	80.00 MiB	---	---	boot
/dev/mmcblk0p4 ⚠ ⬛	unknown	14.76 GiB	---	---	
unallocated	▨ unallocated	14.28 GiB	---	---	

Above: *drive /dev/mmcblk0 is the eMMc storage with a RISC OS filecore partition containing a Loader partition with the firmware for start up.*

Below: *drive /dev/sda is a 32GB pen drive containing the standard Linux distro*

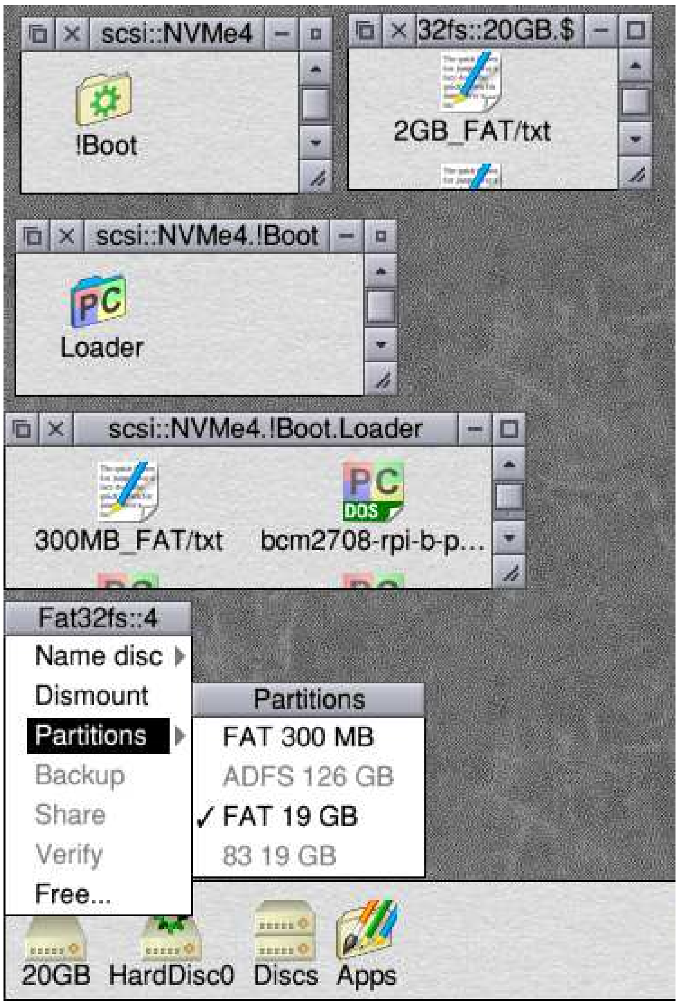

/dev/sda - GParted

GParted Edit View Device Partition Help

/dev/sda (28.65 GiB)

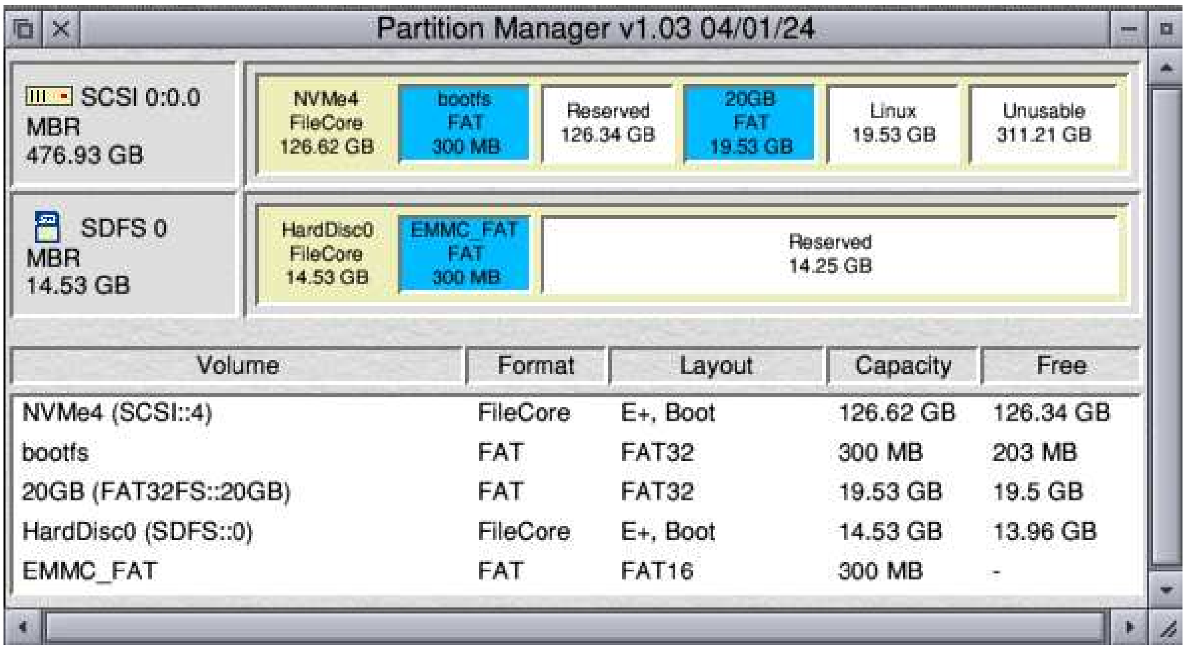
/dev/sda2 9.77 GiB | unallocated 18.62 GiB

Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
unallocated	▨ unallocated			4.00 MiB	---	---	
/dev/sda1	🟩 fat32	/boot	bootfs	256.00 MiB	51.43 MiB	204.57 MiB	lba
/dev/sda2	🟦 ext4	/	rootfs	9.77 GiB	8.82 GiB	973.28 MiB	
unallocated	▨ unallocated			18.62 GiB	---	---	



Above right: *the icon bar icon marked '20GB' provides access to the filecore partition (using ADJUST) or to one of the two FAT partitions. The partition marked '83' (&83 = 131) is the Linux 'ext4' partition which RISC OS can't see. The next icon is amended to show SDFS::0 with a hard disc icon to reflect the fact that it is non-removable eMMc storage. (See WROCC July 2023 'SDFS Icon + OmniDisc')*

Below: *the view under RISC OS using PartMan of the four-partition NVMe drive in its USB caddy as well as its view of the eMMc storage - the usual two-partition solution but with dual boot added.*


Partition Manager v1.03 04/01/24

SCSI 0:0.0 MBR 476.93 GB: NVMe4 FileCore 126.62 GB | bootfs FAT 300 MB | Reserved 126.34 GB | 20GB FAT 19.53 GB | Linux 19.53 GB | Unusable 311.21 GB

SDFS 0 MBR 14.53 GB: HardDisc0 FileCore 14.53 GB | EMMC_FAT FAT 300 MB | Reserved 14.25 GB

Volume	Format	Layout	Capacity	Free
NVMe4 (SCSI::4)	FileCore	E+, Boot	126.62 GB	126.34 GB
bootfs	FAT	FAT32	300 MB	203 MB
20GB (FAT32FS::20GB)	FAT	FAT32	19.53 GB	19.5 GB
HardDisc0 (SDFS::0)	FileCore	E+, Boot	14.53 GB	13.96 GB
EMMC_FAT	FAT	FAT16	300 MB	-

We then prepared an NVMe drive with four partitions by using a USB caddy. Linux now boots from the NVMe drive (in its USB caddy) and RISC OS can 'see' the filecore partition on the NVMe drive (in its USB caddy).

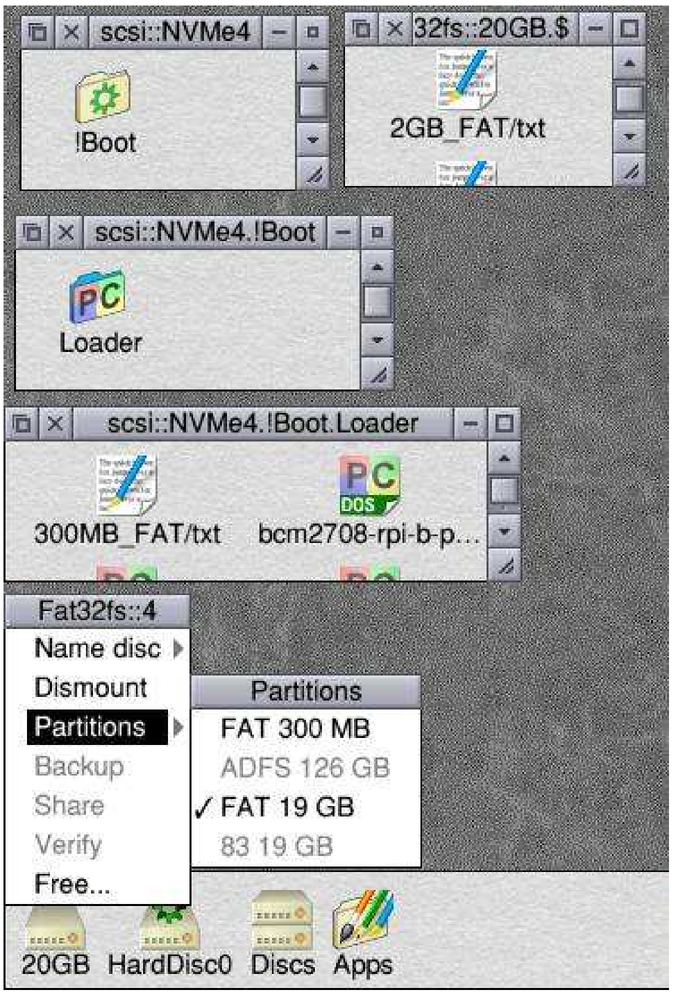
Now we can take the NVMe drive out of the caddy and plug it in to the M.2 NVMe socket directly. Now Linux can still see it and boot from it (with the switch 'ON') using the eMMc storage only to be told where to find its root filesystem (on the NVMe drive).

RISC OS cannot yet see the NVMe drive but once a RISC OS driver exists, it will be able to use the NVMe storage as its boot drive and will then just use the eMMc storage to load the rom.

Conclusion

The bright idea of having a four partition NVMe drive seems to work. Holding it in a USB caddy allows RISC OS to see both FAT partitions and the filecore partition. Once an NVMe driver exists for RISC OS it should work at full speed in the PCIe slot on the Pi Foundation IO board or in its M.2 socket on the Waveshare IO board.

Chris Hall *chris@svrsig.org*



Above right: *The USB caddy that holds the NVMe drive - this makes it visible under RISC OS, but only at USB2 speeds. Placed in a PCIe slot, it will run at full speed under Linux.*

Right: *the NVMe drive is in a USB caddy but a similar view would be obtained once an NVMe driver for RISC OS is available.*