Gadgets USB composites sur le Raspberry Pi Zero

Table of Contents

- Gadgets USB composites sur le Raspberry Pi Zero
  - Présentation de libComposite
  - configuration de la carte SD
  - Configuration du ou des gadgets
    - ConfigFS
    - Créer le script de configuration
  - Création du gadget
    - Adaptateur série
    - Adaptateur Ethernet
    - Clavier/Souris/Joystick (HID)
    - Stockage de masse
    - Périphérique réseau RNDIS

Généralement pour configurer un gadget **USB-OTG** sur Raspberry Pi on ajoute à `/boot/cmdline.txt`, après « rootwait », ceci : `rootwait modules-load=dwc2 ,g_ether` (ou éventuellement `rootwait modules-load=dwc2,g_ether g_ether.host_addr=00:00:5e:00:53:01 g_ether.dev_addr=00:00:5e:00:53:02` qui définit les adresses MAC de chacun fin de la connexion), en plus d'indiquer `dtoverlay=dwc2` dans `/boot/config.txt`.

Bien que cela fonctionne, on ne peut pas charger en même temps un port série (pour « Accès à la console ») ou un volume de stockag.

Lorsque l'on souhaite utiliser l'interface USB pour faire apparaître également un port série (pour « Accès à la console ») ou un volume de stockage alors on ne peut pas les charger en même temps avec les pilotes de gadget (**g_xxxxxx**). **LibComposite** résout ces problèmes en plaçant la configuration dans l'espace utilisateur (**ConfigFS**).

Présentation de libComposite

Dans le noyau Linux, un module appelé «**libcomposite**» est disponible pour créer un périphérique composite **USB-OTG**, qui remplira plusieurs « fonctions » sur la même interface.

Essentiellement, on charge le module du noyau, qui ajoute un nouveau répertoire au système de fichiers « configfs » dans `/sys/kernel/config/usb_gadget`. Ici, on peut créer un nouveau périphérique, puis cela crée une série de chemins pour vous, y compris les fichiers **idVendor**, **idProduct**, **bcdDevice**, **bcdUSB** et **bDeviceClass**. Essentiellement, cela indique au système d'exploitation à quel type d'appareil le port USB est connecté .

Pour que le système d'exploitation sache comment appeler l'appliance que l'on ajoute au port USB, il y a un autre répertoire ici : des chaînes qui contiennent une série de répertoires numérotés hexadécimaux. Basé sur la documentation Microsoft, 0x409 correspond aux chaînes en anglais. Ainsi, dans le chemin strings/0x409/, il y aura trois fichiers, le numéro de série, le produit et le fabricant, qui contiennent la façon dont le périphérique est décrit sur le système d'exploitation. Plutôt que de générer mon propre numéro de série ou de concocter ma propre chaîne de produits, on peut utiliser le numéro de série et les détails du modèle intégrés au processeur, que l'on trouvera dans `/proc/cpuinfo`.

Une fonction, en termes de **libcomposite**, est une fonctionnalité que le périphérique indique à l'hôte qu'il peut exécuter, comme être un adaptateur série, ou un périphérique de stockage, ou un adaptateur réseau. Chaque groupe de fonctions est ajouté à un répertoire de configurations et possède un répertoire de fonctions (comme rndis.usb0 ou ecm.usb0) lié à ce répertoire de configurations. Le répertoire des fonctions contient certaines valeurs de configuration et le répertoire des configurations permet au système d'exploitation de choisir entre plusieurs fonctions (mais généralement une seule), en fonction de certaines valeurs qui y sont stockées.

Chaque « fonction » doit également savoir si l'ordinateur hôte doit l'alimenter, ainsi que la quantité d'énergie dont il a besoin. Ceux-ci sont stockés respectivement dans le fichier `configs/c.<id>/bmAttributes` et dans le fichier `configs/c.<id>/MaxPower`.

configuration de la carte SD

Télécharger et installer la dernière version de Raspbian sur une carte SD suffisamment grande et développer la partition racine.

Maintenant, il faut activer une « superposition d'arborescence de périphériques » spéciale :

```
echo "dtoverlay=dwc2" | sudo tee -a /boot/config.txt
écho "dwc2" | sudo tee -a /etc/modules
```

Enfin, il faut activer le pilote libcomposite en exécutant :

```
sudo echo "libcomposite" | sudo tee -a /etc/modules
```

Configuration du ou des gadgets

Il faut maintenant définir ce que le gadget doit faire – Ethernet ? un clavier? tout ce qui précède?

La configuration se fait via **ConfigFS**, un système de fichiers virtuel dans `/sys/`, qui est automatiquement monté sur le Pi au démarrage.

ConfigFS

La prise en charge de Linux Configurationfs (**CONFIG_CONFIGFS_FS**) permet une configuration

dynamique complète des périphériques gadgets à partir de l'espace utilisateur, auquel cas on peut créer une configuration unique ou un périphérique composite multi-configuration avec une ou plusieurs des fonctions disponibles dans `drivers/usb/gadget/udc/functions` :

- **usb_f_acm**: CDC Serial (ACM - Modèle de contrôle abstrait)
- **usb_f_ecm**: CDC Ethernet (ECM - Modèle de contrôle de réseau Ethernet)
- **usb_f_eem**: CDC Ethernet (EEM - Modèle d'émulation Ethernet)
- **usb_f_fs**: Système de fichiers
- **usb_f_hid**: Interface HID
- **usb_f_mass_storage**: Classe de stockage de masse USB
- **usb_f_midi**: MIDI
- **usb_f_ncm**: Réseau CDC (NCM - Modèle de contrôle réseau Ethernet)
- **usb_f_obex**: CDC OBEX (modèle d'échange d'objets)
- **usb_f_phonet**: CDC Phonet
- **usb_f_printer**: Fonction d'imprimante
- **usb_f_rndis**: (Spécification de l'interface du pilote réseau distant - Microsoft Ethernet sur USB)
- **usb_f_serial**: Fonction série générique
- **usb_f_subset**: Sous-ensemble CDC (Ethernet sans mécanisme de contrôle - juste un transfert de données brutes)
- **usb_f_uac1**: Classe audio USB
- **usb_f_uac2**: Classe audio USB 2.0
- **usb_f_uvc**: Classe vidéo USB

Tous les modules de noyau ci-dessus peuvent ne pas être disponibles en fonction de la configuration dunoyau ou du BSP.

Créer le script de configuration

La configuration est volatile, elle doit donc être exécutée à chaque démarrage.

Créer un fichier `isticktoit_usb` dans `/usr/bin/`. Taper ce qui suit :

```
sudo touch /usr/bin/isticktoit_usb #créer le fichier
sudo chmod +x /usr/bin/isticktoit_usb #le rendre exécutable
sudo vi /usr/bin/isticktoit_usb #modifier le fichier
```

Ensuite, il faut exécuter ce script automatiquement au démarrage. Pour de meilleures performances, on peut créer un fichier d'unité systemd, mais pour l'instant, on s'en tient à rc.local. (cela fait partie de l'ancien système sysvinit, mais est toujours exécuté sur le pi par défaut)

Ouvrir `/etc/rc.local` en tant que root et ajouter la ligne suivante avant (!!!!) la ligne contenant le mot « exit ».

```
sudo vi /etc/rc.local
/etc/rc.local...
/usr/bin/isticktoit_usb # configuration libcomposite
```

```
exit
```

Création du gadget

Il s'agit de la configuration globale ; le nombre de fonctionnalités USB que le Pi utilisera n'a donc pas d'importance. Modifier le numéro de série, le fabricant et le nom du produit dans ce bloc.

```
/usr/bin/isticktoit_usb
#!/bin/bash
cd /sys/kernel/config/usb_gadget/
mkdir -p isticktoit
cd isticktoit
echo 0x1d6b > idVendor # Linux Foundation
echo 0x0104 > idProduct # Multifunction Composite Gadget
echo 0x0100 > bcdDevice # v1.0.0
echo 0x0200 > bcdUSB # USB2
mkdir -p strings/0x409
echo "fedcba9876543210" > strings/0x409/serialnumber
echo "Tobias Girstmair" > strings/0x409/manufacturer
echo "iSticktoit.net USB Device" > strings/0x409/product
mkdir -p configs/c.1/strings/0x409
echo "Config 1: ECM network" > configs/c.1/strings/0x409/configuration
echo 250 > configs/c.1/MaxPower
# Add functions here
# see gadget configurations below
# End functions
ls /sys/class/udc > UDC
```

Adaptateur série

Ceci est très utile pour le débogage et est le plus simple à configurer. dans le fichier `/usr/bin/isticktoit_usb`, insérer ce qui suit entre les commentaires « fonctions » :

```
# Ajouter des fonctions ici
mkdir -p functions/acm.usb0
ln -s functions/acm.usb0 configs/c.1/
# Fin des fonctions
```

Ensuite, activer une console sur la série USB :

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

Pour se connecter au Pi sur un ordinateur Linux, il faut que **screen** soit installé (`sudo apt-get install screen`), puis exécuter :

```
screen sudo /dev/ttyACM0 115200
```

si l'écran se termine automatiquement, il faut modifier le fichier de l'appareil. Consulter par exemple dmesg :

```
dmesg|grep "USB ACM"
```

Adaptateur Ethernet

Tout d'abord, ajouter au fichier /usr/bin/isticktoit_usb:

```
# Ajouter des fonctions ici
mkdir -p fonctions/ecm.usb0
# le premier octet de l'adresse doit être pair
HOST="48:6f:73:74:50:43" # "HostPC"
SELF="42:61:64:55:53:42" # "BadUSB"
echo $HOST > functions/ecm.usb0/host_addr
echo $SELF > functions/ecm.usb0/dev_addr
ln -s functions/ecm.usb0 configs/c.1/
# Fin des fonctions
ls /sys/class/udc > UDC
#mettre ceci à la toute fin du fichier :
ifconfig usb0 10.0.0.1 netmask 255.255.255.252 up
route add -net default gw 10.0.0.2
```

Enregistrer et quitter, puis sur le PC hôte :

Si on rencontre des problèmes avec la connexion automatique (par exemple « Connexion filaire 2 »), se déconnecter et exécuter ceci :

```
dmesg|grep cdc_ether
[13890.668557] cdc_ether 1-1:1.2 eth0 : enregistrer 'cdc_ether' sur
usb-0000:00:14.0-1, périphérique Ethernet CDC, 48:6f:73:74:50:43
[13890.674117] usbcore : nouveau pilote d'interface enregistré cdc_ether
[13890.687619] cdc_ether 1-1:1.2 enp0s20u1i2 : renommé de eth0
```

Cela indique comment l'adaptateur Ethernet est nommé (et peut-être renommé par la suite. Utiliser le nom de l'interface (enp0s20u1i2) dans cette ligne :

```
sudo ifconfig enp0s20u1i2 10.0.0.2 netmask 255.255.255.252 up
```

Ensuite, se connecter via ssh au Pi :

```
ssh 10.0.0.1 -lpi
```

Clavier/Souris/Joystick (HID)

Tout d’abord, ajouter au fichier /usr/bin/isticktoit_usb:

```
# Ajouter des fonctions ici
mkdir -p functions/hid.usb0
echo 1 > functions/hid.usb0/protocol
echo 1 > functions/hid.usb0/subclass
echo 8 > functions/hid.usb0/report_length
echo -ne
\\x05\\x01\\x09\\x06\\xa1\\x01\\x05\\x07\\x19\\xe0\\x29\\xe7\\x15\\x00\\x25\
\x01\\x75\\x01\\x95\\x08\\x81\\x02\\x95\\x01\\x75\\x08\\x81\\x03\\x95\\x05\\
x75\\x01\\x05\\x08\\x19\\x01\\x29\\x05\\x91\\x02\\x95\\x01\\x75\\x03\\x91\\x
03\\x95\\x06\\x75\\x08\\x15\\x00\\x25\\x65\\x05\\x07\\x19\\x00\\x29\\x65\\x8
1\\x00\\xc0 > functions/hid.usb0/report_desc
ln -s functions/hid.usb0 configs/c.1/
# End functions
```

Le moyen le plus simple d'envoyer des frappes au clavier consiste à renvoyer les paquets HID vers le fichier du périphérique :

```
sudo su
echo -ne "\0\0\x4\0\0\0\0\0" > /dev/hidg0 #appuyer sur le bouton A
echo -ne "\0\0\0\0\0\0\0\0" > /dev/hidg0 #libérer toutes les clés
```

Ce n'est cependant pas réalisable, alors rendez-vous sur ma page GitHub et télécharger le code sur votre ordinateur. extraire sur la carte SD du Pi et démarrez-le. Sur le Pi : CD senHID/ make #compiler le programme echo -n “Bonjour tout le monde !” | sudo ./scan /dev/hidg0 1 2

la dernière ligne écrira la chaîne transmise via le protocole HID. Le « 1 » signifie la disposition américaine, un « 2 » à la place serait la disposition allemande/autrichienne. Le deuxième chiffre sert à saisir des caractères non disponibles sur votre clavier (2=Linux, 3=Windows (mais pas de pilotes Windows))

Stockage de masse

Le stockage de masse est quelque peu difficile. Créer un disque. C'est un processus assez long :

```
dd if=/dev/zero of=~/usbdisk.img bs=1024 count=1024
mkdosfs ~/usbdisk.img
```

Ensuite, ajouter au fichier /usr/bin/isticktoit_usb:

```
# Ajouter des fonctions ici
FILE=/home/pi/usbdisk.img
```

```
mkdir -p ${FILE/img/d}
# mount -o loop,ro,offset=1048576 -t ext4 $FILE ${FILE/img/d} # FOR OLD WAY
OF MAKING THE IMAGE
mount -o loop,ro, -t vfat $FILE ${FILE/img/d} # FOR IMAGE CREATED WITH DD
mkdir -p functions/mass_storage.usb0
echo 1 > functions/mass_storage.usb0/stall
echo 0 > functions/mass_storage.usb0/lun.0/cdrom
echo 0 > functions/mass_storage.usb0/lun.0/ro
echo 0 > functions/mass_storage.usb0/lun.0/nofua
echo $FILE > functions/mass_storage.usb0/lun.0/file
ln -s functions/mass_storage.usb0 configs/c.1/
# End functions
```

Un lecteur amovible formaté en FAT32 devrait apparaître la prochaine fois que vous connecterer le Pi à un ordinateur. Pour accéder aux fichiers stockés sur l'image disque à partir du Pi, on peutla démonter complètement (dans l'hôte d'abord, puis dans le pi) et la remonter ailleurs.

Périphérique réseau RNDIS

Avec libcomposite, les développeurs USB ont créé une couche réseau appelée «**ECM** » (ou « Ethernet Control Model) dans le cadre du groupe d'interfaces «**CDC**» (ou « Communications Device Class »). Pour créer un périphérique **ECM**, il faut créér `/sys/kernel/config/usb_gadget/libComposite/functions/ecm.usb0` puis lier tout ce répertoire dans `/sys/kernel/config/usb_gadget/libComposite/configs/c.1/`.

Par défaut, **ECM** fournira une adresse MAC dynamique à la fois pour l'hôte (la machine à laquelle il est connecté) et pour le périphérique (le périphérique branché), à moins que ceux-ci ne soient spécifiés en mettant des valeurs dans `Functions/ecm.usb0/host_addr` et `Functions/ecm.usb0/dev_addr` respectivement.

Mais pour que cela fonctionne sous Windows il faut définir des chaînes magiques dans des fichiers supplémentaires, créer un répertoire appelé `os_desc/interface.rndis` et y coller deux fichiers appelés `compatibility_id` et `sub_compatibility_id`, qui contiennent les chaînes **RNDIS** et **5162001**, il faut également mettre la chaîne **RNDIS** dans `configs/c.2/strings/0x409/configuration`.</WRAP>

Ainsi, une fois les valeurs de configuration chargées, il faut maintenant indiquer au système d'exploitation Raspbian d'utiliser ces valeurs avec le système d'exploitation hôte. Les valeurs sont chargées dans le répertoire `/sys/class/udc` dans un fichier sous le chemin de configuration du gadget **libComposite-UDC**. Et puis le système d'exploitation propose cela au système d'exploitation hôte.

Pour annuler cela pour une raison quelconque, il suffit d'écrire une chaîne vide sur ce fichier UDC et libComposite le supprimera du système hôte.

Cependant rien ne stocke ces valeurs entre les démarrages, il faut donc exécuter un script à chaque démarrage, afin qu'il se propose au système d'exploitation hôte avec une unité de service **SystemD**.

Les unités de service **SystemD** servent essentiellement à indiquer à Linux quels scripts exécuter à chaque démarrage. Il gère également la capture de n'importe quelle sortie de texte et peut démarrer des choses après le démarrage d'autres choses, et ainsi de suite.

Créér un fichier unité `/lib/systemd/system/libComposite.service`, puis le lier à `/etc/systemd/system/multi-user.target.wants` (ce que fait l'exécution de systemctl activate libComposite.service)

```
[Unit]
Description=Start libComposite
Before=dnsmasq.Service

[Service]
Type=oneshot
ExecStart=/opt/lib_composite.sh

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

Fondamentalement, cela dit « exécuter le script dans `/opt/lib_composite.sh` chaque fois que ce service est invoqué », et cela dit également « s'assurer qu'il est exécuté avant de dire au système d'exploitation qu'il est démarré et prêt à ce que les utilisateurs se connectent ».

Si on démarre le système uniquement avec cela, on se retrouvera avec deux interfaces sans configuration - `/dev/usb0` et `/dev/usb1`. Ce n’est pas bon, il faut leur donner des adresses IP.

Configuration des interfaces avec /etc/network/interfaces

Ainsi, parce qu'on a défini l'interface **ECM/CDC** comme **c.1** et l'interface **RNDIS** comme **c.2**, l'interface **ECM** sera **usb0** et **RNDIS usb1**. On peut utiliser **NetworkManager** ou **NetPlan**, ou utiliser le fichier `/etc/network/interfaces`.

On peut mettre un fichier par interface dans `/etc/network/interfaces.d/<interface name>`, donc le premier que l'on va créer est `/etc/network/interfaces.d/usb0` :

```
allow-hotplug usb0
iface usb0 inet static
  address 192.0.2.1
  netmask 255.255.255.240 # soit /28 ou 14 adresses utilisables
```

Créer le même fichier pour usb1 mais définir l'adresse sur 192.0.2.17 à la place.

Maintenant, si on devions démarre ceci, l’extrémité Pi obtiendrait une adresse IP, mais pas l’hôte. Pour résoudre CELA, il faut un serveur **DHCP**, et **DNSMasq** est parfait pour cela.

DNSMasq

DNSMasq est une application d'une simplicité trompeuse permettant d'offrir un service DHCP et DNS. il faut configurer cela à deux endroits, `/etc/dnsmasq.conf` et `/etc/default/dnsmasq`.

Dans la configuration principale `/etc/dnsmasq.conf`, lui indiquer quelles interfaces utiliser, comme ceci :

```
interface=usb0
interface=usb1
bind-interfaces
```

il faut également lui indiquer quelle plage d'adresses IP servir :

```
dhcp-range=usb0,192.0.2.2,192.0.2.14,2h
dhcp-range=usb1,192.0.2.18,192.0.2.30,2h
```

Et enfin, quel itinéraire par défaut servir, car alors que les vitesses de transfert USB sont probablement assez rapides sur mon réseau pour une navigation générale, les connexions réseau WiFi ou filaire sont plus rapides, il faut donc annoncer qu'il n'y a pas de route par défaut pour cette connexion, comme ceci :

```
dhcp-option=option:router
```

Si on veut annoncer les routes par défaut vers cet appareil, on peut conseiller chaque route comme ceci :

```
dhcp-option=usb0,option:router,192.0.2.1
dhcp-option=usb1,option:router,192.0.2.18
dhcp-option=usb0,option:router,192.0.2.1
```

La dernière chose à faire est de dire à **DNSMasq** de ne pas se lier à l'interface « lo » (« loopback »), car on utilisera toujours le service **systemd-resolve**, qui exécute un service DNS sur 127.0.0.53. il faut donc éditer `/etc/default/dnsmasq` et ajouter cette ligne :

```
DNSMASQ_EXCEPT=lo
```

Cela signifie "se lier à toutes les interfaces, à l'exception de l'interface appelée lo".

Et maintenant pour démarrer **DNSMasq**, il faut contrôler quand **DNSMasq** démarre au démarrage, car il repose sur la présence de ces interfaces **usb0** et **usb1**. Alors maintenant, il faut faire quelques changements !

Sur **Raspbian**, lorsqu'on installe **dnsmasq**, il place un fichier d'unité **systemd**, mais il crée également un script d'initialisation « SysV », et le fichier d'unité systemd appelle le fichier **SysV**. Donc lorsque **systemctl** active **dnsmasq** il va trouver le fichier **SysV** `/etc/init.d/dnsmasq`, et l'utilisera de préférence au fichier `/lib/systemd/system/dnsmasq.service`. Il faut donce, tout d'abord, d'arrêter `/etc/init.d/dnsmasq`, le renommer en quelque chose d'autre, puis indiquer à `/lib/systemd/system/dnsmasq.service` d'utiliser le script **init.d** renommé, puis l'activer. Le

script `package_postinstall.sh`, que l'on appelera à partir d'un fichier SystemD Service Unit, permet de faire tout cela.

Configurer une connexion WiFi

Afin d'installer DNSMasq, ainsi que de faire tout ce que il faut faire sur cet appareil nécessitant un réseau, il faut créer un fichier `wpa_supplicant` dans `/boot`. Celui-ci doit stocker le **SSID** et le **PSK** (« clé pré-partagée ») pour le point d'accès WiFi. Ce fichier ressemble à ceci :

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB

network={
  ssid="exampleSSID"
  psk="abc123def456"
}
```

Au premier démarrage, ce fichier sera déplacé au bon endroit sur le système d'exploitation (`/etc/wpa_supplicant/wpa_supplicant.conf`) et supprimé du répertoire `/boot`.

Service SSH

Lors des versions initiales, on a besoin d'un serveur SSH pour pouvoir s'assurer que l'on dispose de tout ce dont on a besoin. Pour ce faire, mettre un fichier SSH dans `/boot`, comme pour le fichier `wpa_supplicant`. Ce fichier peut cependant être vide.

Ensemble des scriptes

- dnsmasq

```
ENABLED=1
CONFIG_DIR=/etc/dnsmasq.d,.dpkg-dist,.dpkg-old,.dpkg-new
IGNORE_RESOLVCONF=yes
DNSMASQ_EXCEPT=lo
```

- installPackages.service

```
[Unit]
Description=Install Required Packages
ConditionPathExists=/root/packages
After=network-online.target

[Service]
Type=oneshot
```

```
ExecStart=/bin/bash -c "[ -x /root/package_preinstall ] &&
/root/package_preinstall && rm -f /root/package_preinstall &&
/opt/installPackages.sh && [ -x /root/package_postinstall ] &&
/root/package_postinstall && rm -f /root/package_postinstall"

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

- installPackages.sh

```
#!/bin/bash
export DEBIAN_FRONTEND=noninteractive
until apt update
do
  sleep 1
done
until apt install -y -o Dpkg::Options::="--force-confdef" -o
Dpkg::Options::="--force-confold" $(cat /root/packages 2>/dev/null)
do
  sleep 1
done
rm -f /root/packages
```

- libComposite.service

```
[Unit]
Description=Start libComposite
Before=dnsmasq.Service

[Service]
Type=oneshot
ExecStart=/opt/lib_composite.sh

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

- libComposite.sh

```
#!/bin/bash
# Based on a combination of: http://www.isticktoit.net/?p=1383
#   and: https://www.raspberrypi.org/forums/viewtopic.php?t=260107
#   and:
https://gist.github.com/schlarpc/a327d4aa735f961555e02cbe45c11667/c80d8894da
6c716fb93e5c8fea98899c9aab8d89
#   and:
https://github.com/ev3dev/ev3-systemd/blob/02caecff9138d0f4dcfeb5afbee67a0bb
689cec0/scripts/ev3-usb.sh

configfs="/sys/kernel/config/usb_gadget"
this="${configfs}/libComposite"
```

```
# Configure values
serial="$(grep 'Serial' /proc/cpuinfo | head -n 1 | sed -E -e
's/^Serial\s+:\s+0000(.+)/\1/')"
model="$(grep 'Model' /proc/cpuinfo | head -n 1 | sed -E -e
's/^Model\s+:\s+(.+)/\1/')"
manufacturer="Raspberry Pi Foundation"

# The serial number ends in a mac-like address. Let's use this to build a
MAC address.
#   The first binary xxxxxx10 octet "locally assigned, unicast" which means
we can avoid
#   conflicts with other vendors.
mac_base="$(echo "${serial}" | sed 's/\(\w\w\)/:\1/g' | cut -b 4-)"
ecm_mac_address_dev="02${mac_base}"  # ECM/CDC address for the Pi end
ecm_mac_address_host="12${mac_base}" # ECM/CDC address for the "host" end
that the Pi is plugged into
rndis_mac_address_dev="22${mac_base}"  # RNDIS address for the Pi end
rndis_mac_address_host="32${mac_base}" # RNDIS address for the "host" end
that the Pi is plugged into

# Make sure that libComposite is loaded
libcomposite_loaded="$(lsmod | grep -e '^libcomposite' 2>/dev/null)"
[ -z "${libcomposite_loaded}" ] && modprobe libcomposite
while [ ! -d "${configfs}" ]
do
  sleep 0.1
done

# Make the path to the libComposite device
mkdir -p "${this}"

echo "0x0200"                     > "${this}/bcdUSB"       # USB Version
(2)
echo "0x1d6b"                     > "${this}/idVendor"     # Device Vendor:
Linux Foundation
echo "0x0104"                     > "${this}/idProduct"    # Device Type:
MultiFunction Composite Device
echo "0x02"                       > "${this}/bDeviceClass" # This means it
is a communications device

# Device Version (this seems a bit high, but OK)
# This should be incremented each time there's a "breaking change" so that
it's re-detected
# rather than cached (apparently)
echo "0x4000"                     > "${this}/bcdDevice"

# "The OS_Desc config must specify a valid OS Descriptor for correct driver
selection"
#   See:
https://www.kernel.org/doc/Documentation/ABI/testing/configfs-usb-gadget
mkdir -p "${this}/os_desc"
```

```
echo "1"                            > "${this}/os_desc/use"          #
Enable OS Descriptors
echo "0xcd"                          > "${this}/os_desc/b_vendor_code" #
Extended feature descriptor: MS
echo "MSFT100"                       > "${this}/os_desc/qw_sign"      # OS
String "proper"

# Configure the strings the device presents itself as
mkdir -p "${this}/strings/0x409"
echo "${manufacturer}"               > "${this}/strings/0x409/manufacturer"
echo "${model}"                      > "${this}/strings/0x409/product"
echo "${serial}"                     > "${this}/strings/0x409/serialnumber"

# Set up the ECM/CDC and RNDIS network interfaces as
#   configs/c.1 and configs/c.2 respectively.
for i in 1 2
do
  mkdir -p "${this}/configs/c.${i}/strings/0x409"
  echo "0xC0"                        > "${this}/configs/c.${i}/bmAttributes" #
Self Powered
  echo "250"                         > "${this}/configs/c.${i}/MaxPower"    #
250mA
done

# Add the Serial interface
mkdir -p "${this}/functions/acm.usb0"
ln -s "${this}/functions/acm.usb0"   "${this}/configs/c.1/"

# Set up the ECM/CDC function
mkdir -p "${this}/functions/ecm.usb0"
echo "${ecm_mac_address_host}"     > "${this}/functions/ecm.usb0/host_addr"
echo "${ecm_mac_address_dev}"      > "${this}/functions/ecm.usb0/dev_addr"
echo "CDC"                         >
"${this}/configs/c.1/strings/0x409/configuration"
ln -s "${this}/functions/ecm.usb0"   "${this}/configs/c.1/"

mkdir -p "${this}/functions/rndis.usb0"
mkdir -p "${this}/functions/rndis.usb0/os_desc/interface.rndis"
echo "RNDIS"                       >
"${this}/functions/rndis.usb0/os_desc/interface.rndis/compatible_id"
echo "5162001"                     >
"${this}/functions/rndis.usb0/os_desc/interface.rndis/sub_compatible_id"
echo "${rndis_mac_address_host}"   >
"${this}/functions/rndis.usb0/host_addr"
echo "${rndis_mac_address_dev}"    > "${this}/functions/rndis.usb0/dev_addr"
echo "RNDIS"                       >
"${this}/configs/c.2/strings/0x409/configuration"
ln -s "${this}/configs/c.2"          "${this}/os_desc"
ln -s "${this}/functions/rndis.usb0" "${this}/configs/c.2/"

udevadm settle -t 5 || true
```

```
ls /sys/class/udc > "${this}/UDC"

systemctl start getty@ttyGS0.service
```

- setPiPassword.service

```
[Unit]
Description=Set password for the account "pi"
ConditionPathExists=/root/piPassword
After=network-online.target

[Service]
Type=oneshot
ExecStart=/bin/bash -c "cat /root/piPassword | /usr/sbin/chpasswd && rm
/root/piPassword"

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