NUT USB setup in modern Solaris-like systems (OpenSolaris descendants)

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.8.0	2022-12-31	Current release of Network UPS Tools (NUT).	
2.6.0	2011-01-14	First release of AsciiDoc documentation for Network UPS Tools (NUT).	

Contents

Local-media device setup for use with NUT has some nuances with numerous descendants of the OpenSolaris project, including both the commercial Sun/Oracle Solaris 11 and illumos-based open source distributions such as OpenIndiana and OmniOS. Recommendations below may also apply to other related operating systems, possibly to older releases as well.

1 Change the OS driver binding: use UGEN

Like other hardware, USB devices are interfaced to the operating system by OS drivers, and often there are several suitable drivers with different capabilities. In Solaris and related systems, this mapping is detailed in the `/etc/driver_aliases` file and properly managed by dedicated tools. By default, USB devices can be captured by the generic USB HID driver, or none at all; however an "UGEN" driver can behave better with the libusb library used on Solaris.

Note
Operations below would need running as `root` or elevating the privileges (via `pfexec`, `sudo`, etc.)

Connect the power device using its USB port to your computer.

Run `prtconf -v | less` to see the details of device connections, and search for its probable strings (vendor, model, serial number).

Two examples follow:

- In this example, no suitable driver was attached "out of the box":

```
:; prtconf -v
...
 input (driver not attached)
    Hardware properties:
        name='driver-minor' type=int items=1
            value=00000000
        name='driver-major' type=int items=1
            value=00000002
        name='low-speed' type=boolean
        name='usb-product-name' type=string items=1
            value='Eaton 9PX'
        name='usb-vendor-name' type=string items=1
            value='EATON'
        name='usb-serialno' type=string items=1
            value='G202E02032'
        name='usb-raw-cfg-descriptors' type=byte items=34
            value=09.02.22.00.01.01.00.a0.0a ↩
                .09.04.00.00.01.03.00.00.00.09.21.10.01.21.01.22.10.0d ↩
                .07.05.81.03.08.00.14
        name='usb-dev-descriptor' type=byte items=18
            value=12.01.10.01.00.00.00.08.63.04.ff.ff.00.01.01.02.04.01
        name='usb-release' type=int items=1
            value=00000110
        name='usb-num-configs' type=int items=1
            value=00000001
        name='usb-revision-id' type=int items=1
            value=00000100
        name='usb-product-id' type=int items=1
            value=0000ffff
        name='usb-vendor-id' type=int items=1
            value=00000463
        name='compatible' type=string items=9
            value='usb463,ffff.100' + 'usb463,ffff' + 'usbif463,class3.0.0' + 'usbif463, ↩
                class3.0' + 'usbif463,class3' + 'usbif,class3.0.0' + 'usbif,class3.0' + ' ↩
                usbif,class3' + 'usb,device'
```

```
        name='reg' type=int items=1
            value=00000002
        name='assigned-address' type=int items=1
            value=00000003
```

- In the following example, a "hid power" driver was attached, giving some usability to the device although not enough for NUT to interact well (at least, according to the helpful notes in the https://web.archive.org/web/20140126045707/http://-barbz.com.au/blog/?p=407 blog entry):

```
:; prtconf -v
...
 input, instance #1
   Driver properties:
        name='pm-components' type=string items=3 dev=none
            value='NAME= hid1 Power' + '0=USB D3 State' + '3=USB D0 State'
   Hardware properties:
        name='driver-minor' type=int items=1
            value=00000000
        name='driver-major' type=int items=1
            value=00000002
        name='low-speed' type=boolean
        name='usb-product-name' type=string items=1
            value='USB to Serial'
        name='usb-vendor-name' type=string items=1
            value='INNO TECH'
        name='usb-serialno' type=string items=1
            value='20100826'
        name='usb-raw-cfg-descriptors' type=byte items=34
            value ↩
                =09.02.22.00.01.01.03.80.32.09.04.00.00.01.03.00.00.04.09.21.00.01.00.01.22.1 ↩
                b.00.07.05.81.03.08.00.20
        name='usb-dev-descriptor' type=byte items=18
            value=12.01.10.01.00.00.00.08.65.06.61.51.02.00.01.02.03.01
        name='usb-release' type=int items=1
            value=00000110
        name='usb-num-configs' type=int items=1
            value=00000001
        name='usb-revision-id' type=int items=1
            value=00000002
        name='usb-product-id' type=int items=1
            value=00005161
        name='usb-vendor-id' type=int items=1
            value=00000665
        name='compatible' type=string items=9
            value='usb665,5161.2' + 'usb665,5161' + 'usbif665,class3.0.0' + 'usbif665, ↩
                class3.0' + 'usbif665,class3' + 'usbif,class3.0.0' + 'usbif,class3.0' + ' ↩
                usbif,class3' + 'usb,device'
        name='reg' type=int items=1
            value=00000003
        name='assigned-address' type=int items=1
            value=00000005
   Device Minor Nodes:
        dev=(108,2)
           dev_path=/pci@0,0/pci8086,7270@1d/hub@1/input@3:hid_0_1
                spectype=chr type=minor
                dev_link=/dev/usb/hid0
```

You can also check with `cfgadm` if the device is at least somehow visible (if not, there can be hardware issues in play). For example, if there is a physical link but no recognized driver was attached, the device would show up as "unconfigured":

```
:; cfgadm | grep usb-
usb8/1                 usb-input    connected    unconfigured ok
```

If you conclude that a change is needed, you would need to unload the existing copy of the "ugen" driver and set it up to handle the device patterns that you find in *compatible* values from `prtconf`. For example, to monitor the devices from listings above, you would run:

```
:; rem_drv ugen
:; add_drv -i '"usb463,ffff.100"' -m '* 0666 root sys' ugen
```

or

```
:; rem_drv ugen
:; add_drv -i '"usb665,5161.2"' -m '* 0666 root sys' ugen
```

Note that there are many patterns in the *compatible* line which allow for narrower or wider catchment. It is recommended to match with the narrowest fit, to avoid potential conflict with other devices from same vendor (especially if the declared identifiers are for a generic USB chipset).

Also note that the `add_drv` definition above lists the POSIX access metadata for the device node files that would be generated when the device is plugged in and detected. In the examples above, it would be owned by `root:sys` but accessible for reads and writes (`0666`) to anyone on the system. On shared systems you may want to constrain this access to the account that the NUT driver would run as.

After proper driver binding, `cfgadm` should expose the details:

```
# cfgadm -lv
...
usb8/1                         connected    configured   ok
    Mfg: EATON  Product: Eaton 9PX  NConfigs: 1  Config: 0  <no cfg str descr>
    unavailable  usb-input    n        /devices/pci@0,0/pci103c,1309@1d,2:1
...
```

Usually the driver mapping should set up the "friendly" device nodes under `/dev/` tree as well (symlinks to real entries in `/devices/`) so for NUT drivers you would specify a `port=/dev/usb/463.ffff/0` for your new driver section in `ups.conf`.

Note
As detailed in config-notes.txt, the "natively USB" drivers (including `usbhid-ups` and `nutdrv_qx`) match the device by ID and/or strings it reports, and so effectively require but ignore the `port` option — so it is commonly configured as `port=auto`. Drivers used for SHUT or serial protocols do need the device path.

For some serial-to-USB converter chips however it was noted that while the device driver is attached, and the `/device/...` path is exposed in the `dmesg` output (saved to `/var/adm/messages`) the `/dev/...` symlinks are not created. In this case you can pass the low-level name of the character-device node as the "port" option, e.g.:

```
./mge-shut -s 9px-ser -DDDDD -d2 -u root \
    -x port=/devices/pci@0,0/pci103c,1309@1a,2/device@1:0
```

2 libusb version and binary

Until NUT release 2.7.4 the only option to build NUT drivers for USB connectivity was to use libusb-0.1 or a distribution's variant of it; the original Sun Solaris releases and later related systems provided their customized version for example (packaged originally as `SUNWlibusbugen`, `SUNWugen{,u}` and `SUNWusb{,s,u,vc}`).

However, libusb-0.1 consuming programs had some stability issues reported when running with long-term connections to devices (such as an UPS), especially when using USB hubs and chips where hardware vendors had cut a few corners too many, which were addressed in a newer rewrite of the library as libusb-1.0.

Subsequently just as at least the illumos-based distributions evolved to include the new library and certain patches for it, and the library itself matured, the NUT project also added an ability to build with libusb-1.0 either directly or using its 0.1-compat API (available since NUT 2.8.0 release).

If your "standard" build of NUT has problems connecting to your USB UPS (libusb binary variant should be visible in driver debug messages), consider re-building NUT from source with the other option using the recent library build as available for your distribution.

In this context, note the OpenIndiana libusb-1 package pull requests with code which was successfully used when developing this documentation:

- https://github.com/OpenIndiana/oi-userland/pull/5382
- (TO CHECK) https://github.com/OpenIndiana/oi-userland/pull/5277

Binaries from builds made in OpenIndiana using the recipe from PR #5382 above were successfully directly used on contemporary OmniOS CE as well.