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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.8.1	2023-10-31	Some changes to API, docs and recipes, in particular to simplify local builds and tests (e.g. to help end-users check if current NUT codebase trunk has already fixed an issue they see with a packaged installation). Revived NUT for Windows effort, further improved other OS integrations. NUT became reference for "UPS management protocol", Informational RFC 9271. Documentation files refactored to ease maintenance. More drivers and new driver categories introduced.	JK
2.8.0	2022-04-26	Change of maintainer. Many changes to API, docs (both style and content), and recipes, with a stress on non-regression test-ability, run-time debug-ability, general codebase maintainability, as well as OS integrations (notably nut-driver-enumerator for systemd and SMF service instance maintenance). Added a lot in area of CI support and documented pre-requisite package lists for numerous platforms, and CI agent set-up. Added libusb-1.x support and many new driver categories (and drivers), and daisychain device connection support. Instant commands enhanced with TRACKING to enable protocol-based waiting for completion of a particular INSTCMD or SET operation.	JK
2.7.4	2016-03-09	NUT variables namespace updated, in particular for outlet groups, alarms and thresholds, ATS devices, and battery.charger.status to supersede CHRG and DISCHRG flags published in ups.status readings. NUT network protocol extended with NUMBER type; some API changes.	AQ
2.7.3	2015-04-22	Documentation revised, including some API changes. Added NUT DDL links. NUT variables namespace updated.	AQ
2.7.2	2014-04-17	The nut-website project was offloaded into a separate repository. FreeDesktop HAL support was removed (obsoleted in GNOME consumer). Introduced nutdrv_atcl_usb driver.	AQ

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.7.1	2013-11-19	NUT source codebase migrated from SVN to Git (and from Debian infrastructure to GitHub source code hosting). jNut binding split into a separate project. Introduced libnutclient (C++ binding), al175, apcupsd-ups and nutdrv_qx drivers, Mozilla NSS support for simpler licensing than OpenSSL, and a newer apcsmart implementation. Documentation support enhanced with a spell checker, contents massively updated to reflect project changes.	CL
2.6.5	2012-08-08	New macosx-ups driver, new implementation of mge-shut driver. NUT variables namespace updated. Docs cleaned up and revised.	AQ
2.6.4	2012-05-31	New NUT network protocol commands (LIST CLIENTS, LIST RANGE and NETVER), and socket protocol commands (ADDRANGE, DELRANGE). NUT variables namespace updated. Introduced nut-recorder tool.	AQ
2.6.3	2012-01-04	No substantial changes to documentation.	AQ
2.6.2	2011-09-15	Introduced nut-scanner tool and nut-ipmipsu driver, systemd support, and a new apcsmart implementation.	AQ
2.6.1	2011-06-01	Introduced default.* and override.* optional settings in ups.conf, an ups.efficiency report, and outlet.0 special handling.	AQ
2.6.0	2011-01-14	First release of AsciiDoc documentation for Network UPS Tools (NUT).	AQ

Contents

1 Change the OS driver binding: use UGEN **1**

2 libusb version and binary **3**

3 Troubleshooting and reconnecting **4**

3.1 Recycle the USB connection 5

3.2 Regular auto-recovery via crontab 6

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.

3 Troubleshooting and reconnecting

So... your setup worked nicely, and then one day you see the console flooded with messages like the following:

```
Broadcast Message from nut (???) on n54l Mon May  9 12:05:59...
Communications with UPS innotech@localhost lost


Broadcast Message from nut (???) on n54l Mon May  9 12:10:55...
UPS innotech@localhost is unavailable
```

Unfortunately, some devices "get stuck" on USB level (whether in the chips, in the OS driver layer, libusb or NUT driver) and their NUT drivers have to be restarted to regain monitoring, as opposed to intermittent losses of connectivity that software recovers from automatically.

As in all systems, you should stop all programs using the connection, including NUT driver instances that might have been started beside the wrapping service (SMF). It may be possible to just start the new driver instance at this point, but if it still does not see the device — you have to re-initialize the connection on the OS level.

As a symptom, attempts to start the NUT driver with elevated debug verbosity would not even see the device details:

```
   0.000606     [D1] Saving PID 5187 into /var/run/nut/nutdrv_qx-innotech.pid
   0.000727     [D1] upsdrv_initups...
   0.012065     [D2] Checking device 1 of 2 (0665/5161)
   0.012303     [D1] Failed to open device (0665/5161), skipping: Other error
   0.012394     [D2] Checking device 2 of 2 (099A/610A)
...
   0.020364     [D2] Trying to match device
   0.020586     [D3] match_function_regex: matching a device...
   0.020839     [D2] match_function_regex: failed match of VendorID:  99a
   0.021061     [D2] Device does not match - skipping
   0.021371     [D2] libusb1: No appropriate HID device found
Network UPS Tools - Generic Q* USB/Serial driver 0.32 (2.8.0-20-g535395363)
USB communication driver (libusb 1.0) 0.43
   0.021720     libusb1: Could not open any HID devices: insufficient permissions on ↩
       everything
   0.021821     No supported devices found. Please check your device availability with ' ↩
       lsusb'
and make sure you have an up-to-date version of NUT. If this does not help,
try running the driver with at least 'subdriver', 'vendorid' and 'productid'
options specified. Please refer to the man page for details about these options
(man 8 nutdrv_qx).
```

```
Driver failed to start (exit status=1)
Network UPS Tools - UPS driver controller 2.8.0-20-g535395363
[ May  9 03:10:01 Method "start" exited with status 1. ]
```

Note
Details of the service instance life-cycle for the NUT driver may be seen in its SMF log, e.g. by `less /var/svc/log/*innotech*log`, and to see in-vivo debugs as the service starts in production mode, use `debug_min = 3` in the `/etc/nut/ups.conf` file (in global context or in driver section).

3.1 Recycle the USB connection

In case of Solaris/illumos systems, first stop the respective nut-driver instance, e.g.:

```
:; svcadm disable -ts nut-driver:innotech

:; ps -ef | grep -Ei 'nut|ups' ; svcs -p innotech
    root 10522     1   0   May 06 ?           0:00 /usr/sbin/upsmon
    root 16927     1   0   Feb 25 ?           1:20 /usr/lib/nut/bin/nutdrv_qx -a innotech
     nut 10257     1   0   May 06 ?           0:39 /usr/sbin/upsd
    root 16985 15379   0 11:27:36 pts/1       0:00 grep -Ei nut|ups
     nut 10524 10522   0   May 06 ?           0:25 /usr/sbin/upsmon
STATE          STIME    FMRI
offline         11:26:49 svc:/system/power/nut-driver:innotech

# In the ps listing above, a driver daemon is seen that was started as
# the root user beside the actual service. It has to be stopped too:
:; kill -9 16927
```

To unconfigure and disconnect the USB link on the OS level, you will need its attachment point identifier. If you don't know your system's current layout (it can change with device re-enumeration due to hot plugging and/or reboots), you can execute `cfgadm -lv`, look for the "Information" field resembling your UPS brand, and make note of its "Ap_Id". You can also query a single device to confirm a guess or your earlier records:

```
:; cfgadm -lv usb10/1

Ap_Id                          Receptacle   Occupant     Condition
Information
When         Type         Busy     Phys_Id

usb10/1                        connected    configured   ok
Mfg: INNO TECH  Product: USB to Serial  NConfigs: 1  Config: 0  : 20100826
unavailable  usb-input    n        /devices/pci@0,0/pci103c,1609@13:1
```

Disconnect the device; note that if something (typically a program with an open connection) still has a hold on the device, the system would fail to complete the command:

```
:; cfgadm -c disconnect usb10/1

Disconnect the device: /devices/pci@0,0/pci103c,1609@13:1
This operation will suspend activity on the USB device
Continue (yes/no)? yes
cfgadm: Hardware specific failure: Cannot issue devctl
  to ap_id: /devices/pci@0,0/pci103c,1609@13:1
```

If that is the case, run `ps` per above and make sure all NUT driver daemons are stopped (the data server `upsd` and client upsmon should be inconsequential in this regard).

Normally, the reconnection should work like this:

```
:; cfgadm -c unconfigure usb10/1
Unconfigure the device: /devices/pci@0,0/pci103c,1609@13:1
This operation will suspend activity on the USB device
Continue (yes/no)? yes

:; cfgadm -c disconnect usb10/1
Disconnect the device: /devices/pci@0,0/pci103c,1609@13:1
This operation will suspend activity on the USB device
Continue (yes/no)? yes

:; cfgadm -lv usb10/1
Ap_Id                          Receptacle   Occupant     Condition  Information
When         Type         Busy     Phys_Id

usb10/1                        disconnected unconfigured ok
unavailable  unknown      n        /devices/pci@0,0/pci103c,1609@13:1

:; cfgadm -c configure usb10/1
cfgadm: Hardware specific failure: Cannot issue devctl
  to ap_id: /devices/pci@0,0/pci103c,1609@13:1

# Despite the error above, the device is seen now:
:; cfgadm -lv usb10/1
Ap_Id                          Receptacle   Occupant     Condition
Information
When         Type         Busy     Phys_Id

usb10/1                        connected    configured   ok
Mfg: INNO TECH  Product: USB to Serial  NConfigs: 1  Config: 0  : 20100826
unavailable  usb-input    n        /devices/pci@0,0/pci103c,1609@13:1

# ... and the driver can start:
:; svcadm enable innotech
```

When everything gets recovered, you should see it:

```
Broadcast Message from nut (???) on n54l Mon May  9 12:12:30...
Communications with UPS innotech@localhost established
```

and `upsc innotech@localhost` would tell you what it sees :)

3.2 Regular auto-recovery via crontab

Additional tricks that can help involve `crontab` for regular automated checks if the device got lost. One is just an attempt to "clear" the service if its earlier startup failed (repetitively) so SMF gave up:

- * * * * svcadm clear innotech 2>&1 | grep -v *is not in a maintenance*

Another is more complicated and involves some custom scripting:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * MODE=optional /etc/nut/reset-ups-usb- ↩
    solaris.sh
```

... where the script would be a copy (customized to your device(s) and connection points!) of `reset-ups-usb-solaris.sh.samp` from either `scripts/Solaris/` directory in the NUT sources, or a copy which may be available in your system, e.g. under the `/usr/share/nut/solaris-init/` data directory.